CR 115402

Axiomatix

Marina del Rey • California

100

A PARAMETRIC STUDY OF THE COMPLEXITY OF SEQUENTIAL DECODERS

FINAL REPORT
VOLUME II

Contract No.: NAS 9-12091

Prepared by

Gaylord K. Huth

Axiomatix
13900 Panay Way, Suite 110M
Marina del Rey, California 90291

Prepared for

National Aeronautics and Space Administration
Manned Spacecraft Center
Houston, Texas 77058

Axiomatix Report No. R7201-1
27 January 1972

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

VOLUME II

LIST OF TABLES

SECTION 4.0

SEQUENTIAL DECODING

4.1 SEQUENTIAL DECODING AS A TREE SEARCHING ALGORITHM

The sequential decoder accepts the sequence of symbols from the channel denoted by $r = \{r_{i,j}\}$ where the index i refers to the received branch and the index j refers to one of the n symbols on the branch for a code rate $R = 1/n$. From the received sequence, the sequential decoder attempts to find a path $\underline{X} = \{x_{i,j}\}$ through the tree diagram which has a high likelihood of producing the sequence $\underline{r}$. It does so by selecting a tentative path in the tree diagram starting at the all-zeroes state and at each successive node following, the branch (transition) that best matches the appropriate segment of $\underline{r}$. Whenever the path that the decoder is currently following becomes too unlikely, a search is initiated for a better one. To determine the likelihood of a given path in the tree diagram, a metric is established as a measure of how different a particular path is from the received sequence. As an example of how the metric measures the likelihood, consider ideal coherent binary phase-shift-keyed (PSK) modulation over a Gaussian noise channel and eight-level quantization on the demodulated symbols. The signal strength and noise density is normalized such that the noise density has unit variance and the received signal amplitude is $\pm\sqrt{2E_s/N_o}$, where E_s/N_o is the signal energy-to-noise one-sided spectral density ratio of each symbol. In this case, the nearly optimum choice (no noticeable

degradation in performance) of the quantizing thresholds is 0, ± 0.5, ± 1.0, and ± 1.5. Now assign the quantization bits to the eight levels as follows:

111	110	101	100	000	001	010	011
-1.5	-1.0	-0.5	0	0.5	1.0	1.5	

Using this notation, the first quantization bit represents the sign, 0 for + and 1 for - (this is also the convention for encoding PSK with binary symbols), the next two quantization bits represent the magnitude. For a value of E_s/N_o equal to 0 dB, the probability of the assignment of a given quantization to a received symbol is presented in Table 4.1.

Table 4.1 Probability of Quantization Assignment

Symbol Transmitted	Quantization							
	111	110	101	100	000	001	010	011
0	0.00179	0.00613	0.02	0.051	0.102	0.159	0.195	0.465
1	0.465	0.195	0.159	0.102	0.051	0.02	0.00613	0.00179

To compute the n symbol metrics that form the branch metric along a path, each symbol on the branch corresponding to a transmitted zero is "exclusive-OR"-ed (modulo-2 addition) to the sign bit of the received symbol quantization. The metric interval is defined as the result of "exclusive-OR"-ing the symbol on the zero branch with the sign bit of the received symbol quantization. If I_0 is the probability of a given quantization assignment for a transmitted zero and I_1 is the

probability for a transmitted one, then the symbol metric along the zero branch is given by:

$$W_{s0} = c\left[1 - U - \log_2\left(\frac{I_1}{I_0}\right)\right] \tag{4.1}$$

where c is a scale factor and U is the bias (to be optimized but simulations indicate the best choice is equal to the code rate). Since the best codes have generators that tap the input sequence to the convolutional encoder, then the code symbols on the one branch are complements of the symbols on the zero branch and the symbol metric along the one branch is given by:

$$W_{s1} = c\left[1 - U - \log_2\left(\frac{I_0}{I_1}\right)\right] \tag{4.2}$$

If a symbol on the zero path is 1 and the received symbol quantization is 110, the resulting metric interval is 010. From Table 4.1, it is seen that $I_0 = 0.195$ and $I_1 = 0.00613$ for the metric interval 010. Hence, the symbol metric along the zero branch is $W_{s0} = c(5.87-U)$ and along the one branch is $W_{s1} = -c(7.3+U)$.

The branch metric is the sum of the n symbol metrics for code rate 1/n along the branch. The path metric is the sum of the branch metrics over the branches in the path. The bias U is chosen so that the path metric of a long transmitted path (code word) increases with the length of the path while any other long path has a path metric that decreases with the length of the path. The scale factor c is usually

chosen for implementation convenience.

The path metric for a given path $\underline{u}$ in the tree diagram of length h branches is denoted $W(\underline{u}, h)$. In terms of $W(\underline{u}, h)$, the sequential decoder attempts to hypothesize $\underline{u}$ through the tree diagram for which $W(\underline{u}, h)$ increases with h. If $W(\underline{u}, h)$ starts to decrease with increasing h, then the decoder searches back to find a path $\underline{u}'$ for which $W(\underline{u}', h)$ increases. To establish whether a path has an increasing or decreasing metric with h and to eliminate the need to store large values of $W(\underline{u}, h)$, thresholds are established with spacing Δ. After each move forward, $t\Delta$ is subtracted from $W(\underline{u}, h)$ such that $t\Delta \leq W(\underline{u}, h) < (t+1)\Delta$, where t is an integer. Hence, under normal conditions after a move forward and the metric adjustment, $0 \leq W(\underline{u}, h) < \Delta$. If the decoder reaches some node in the path such that each branch forward from the node will make $W(\underline{u}, h)$ negative, then the decoder cannot continue forward at this value of the threshold. Referring to the flow chart in Figure 4.1, it is seen that, if $W(\underline{u}, h)$ is negative along the most probable branch, then the sequential decoder tests the path metric $W(\underline{u}, h-1)$ corresponding to the branch previously followed forward. The branch just behind the branch for which $W(\underline{u}, h)$ was first negative must be positive since $t\Delta$ was subtracted to make $0 \leq W(\underline{u}, h-1) < \Delta$. However, as the decoder backs up, branches farther back may be negative. For example, if $W(\underline{u}, h-2) < W(\underline{u}, h-1) > \Delta$ and at least Δ was subtracted from $W(\underline{u}, h-1)$, then $W(\underline{u}, h-2) < 0$. If the path metric is positive, then paths are

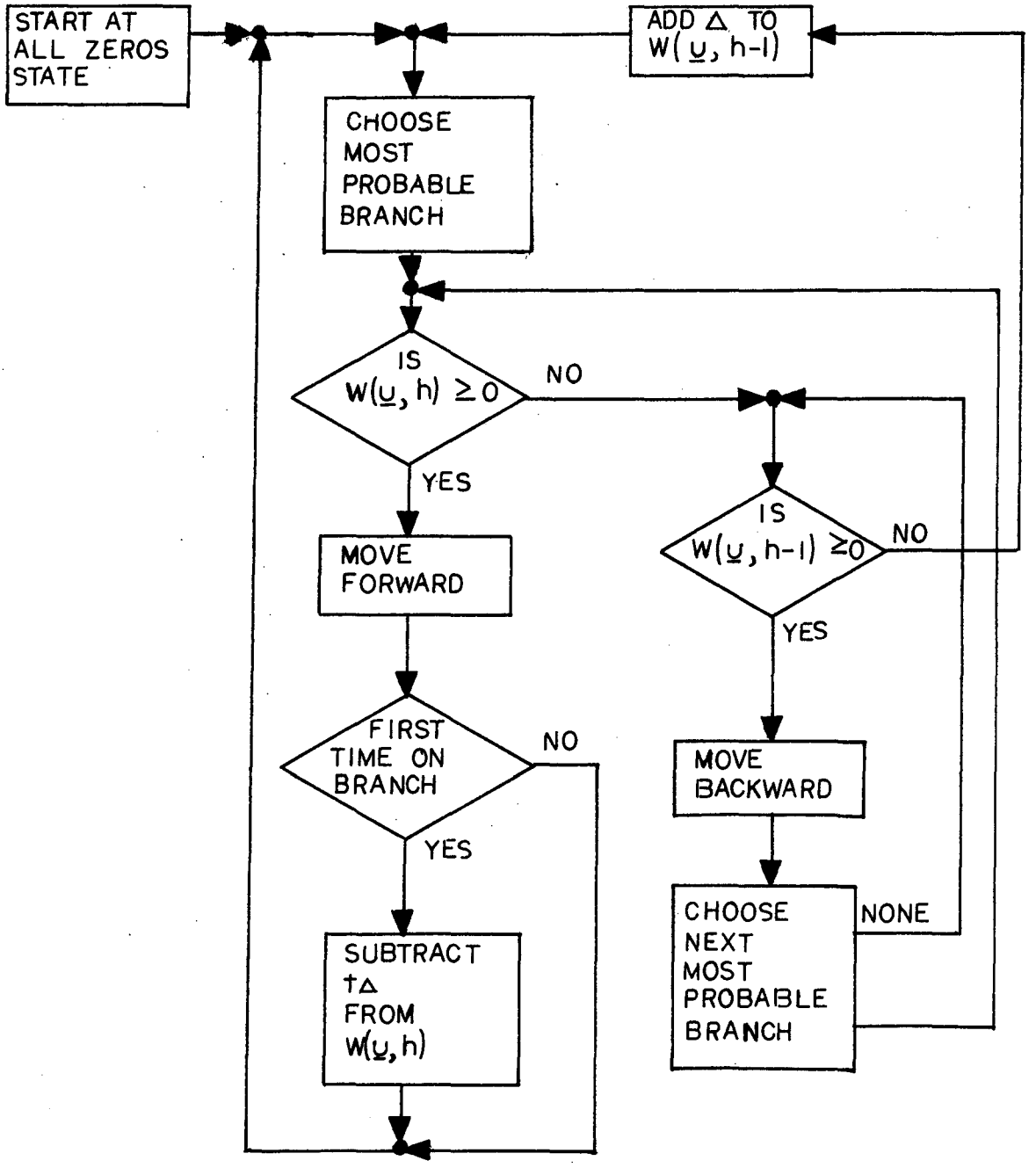


Figure 4.1 Logical Flow Chart of the Sequential Decoding Algorithm

examined leading from the other branch in the binary case (the next most probable branch in any case) if it has not already been examined. If both branches have been examined, then the decoder must back up another branch. If the path metric is negative, then Δ is added to it and the testing continues. If the path metric has been increased by Δ and a branch is re-examined, then the metric must not be allowed to have $t\Delta$ subtracted from it or the sequential decoder could enter a loop.

To more easily understand the operation of the sequential decoder, an example is presented for the rate $R = 1/2$ convolutional code of Figures 2.1 and 2.2. Also, to simplify the example, hard decisions on the demodulated symbols are assumed. For ideal coherent PSK modulation over a Gaussian channel with $E_s/N_o = 1.8$ dB, the probability of error is 0.045. Therefore, $I_0 = 0.955$ and $I_1 = 0.045$ with the symbol metric $w = c(0.933-U)$ for a match between the received symbol and the symbol on the path, while the symbol metric $w' = -c(3.47+U)$ is for a mismatch. A choice commonly used for c and U is $c = 1.36$ and $U = 0.567$ for a rate 1/2 code. In this case, the branch metric is 1 for a double agreement between received symbols and the symbols on the branch, -5 for a single agreement, and -11 for a double disagreement. By simulations, it is found that, using these branch metrics, the best performance is obtained for threshold spacing $\Delta = 9$.

Following the logical flow diagram in Figure 4.1 and the tree diagram in Figure 2.2, the tree searching of the sequential decoder to find the most likely transmitted path for a received sequence of 0101000000 is shown in Table 4.2. The variable θ has not been discussed but its purpose will become clear during the discussion of Table 4.2. In decoding step 1, the received branch (the first two

Table 4.2 Example of the Sequential Decoding Algorithm

Decoding Step	Node	Branch	Decoded Sequence	$W(\underline{u}, h)$	$W(\underline{u}, h-1)$	θ
1	a	00	-	-5	0	0
2	a	00	-	4	9	1
3	aa	00	0	-1	4	0
4	a	11	-	4	9	1
5	ab	00	1	-1	4	0
6	a	00	-	13	18	1
7	aa	00	0	8	13	1
8	aaa	00	00	0	8	0
9	aaaa	00	000	1	0	0
10	aaaaa	00	0000	2	1	0
11	aaaaaa		00000		2	0

symbols) is 01. However, this is only a single agreement no matter which branch emanating from the base node or origin node is examined by the decoder. Hence, there is a tie path. In this case, either branch may be examined first but, by convention, the branch is followed for which the first of the two received symbols matches the first symbol on the branch. Therefore, the branch with symbols 00 is attempted, but since the branch metric is -5 for a single agreement, and is to be

added to the path metric of 0 at the origin, the decoder cannot move forward without making the path metric negative. Following the flow chart for this condition, the decision block "Is $W(\underline{u}, h-1) \geq 0$" is encountered. Normally, since the path metric $W(\underline{u}, h-1) = 0$, the decoder would back up, but at the origin, the decoder cannot back up, so the alternate decision must be made to increase $W(\underline{u}, h-1)$ by Δ, making it equal to 9. Now, the decoder can move forward along the 00 branch, resulting in a path metric equal to 4. When the second branch in the path is attempted, there is another tie branch. Attempting to follow the 00 path results in a path metric of -1. This time, $W(\underline{u}, h-1) = 9$. Therefore, the decoder moves backward and selects the other branch emanating from the origin, the 11 branch. Following the 11 branch from the origin, the path metric becomes 4 since there is a single agreement on the branch. To represent the nodes along the path, a sequence of labels on the tree diagram is used beginning with the origin node labeled a. Thus, from node ab, the decoder attempts the 00 branch. However, this branch also results in a path metric of -1. Therefore, the decoder backs up to the origin node a and, since there are no other untried branches emanating from the origin, Δ is added to the path metric $W(\underline{u}, h-1)$ making it equal to 18. Now, when the 00 branch emanating from the origin is attempted, the path metric becomes 13 and, when the 00 branch emanating from the node aa is attempted, the path metric becomes 8. Beyond the node aa, the received sequence

completely agrees with the uppermost path in the tree diagram. Therefore, the remaining branch metrics are each equal to 1, corresponding to a double agreement branch.

In decoding step 8, the path metric is increased from 8 to 9, but since this is the first time on this branch, $t\Delta$ is subtracted from the path metric with $t = 1$ making the path metric 0. In decoding step 6, the path metric was large enough to have Δ subtracted from it, but since this was not the first time on this branch, it was not allowed. If Δ had been subtracted from the path metric in decoding step 6, then the decoder would follow the same decoding steps as for steps 3-5, and the decoder would be in a loop. If the decision block "FIRST TIME ON BRANCH" was implemented in a brute force method, a very large memory would be necessary. However, it has been proven by Fano[5] that a single binary variable can perform this function. In this case, a variable θ is set equal to 1 each time the decoder backs up. The variable θ is set equal to 0 in the forward path search of the decoder when the path metric is $0 \leq W(\underline{u}, h) < \Delta$. When the variable θ is equal to 1, then the decoder is not allowed to subtract $t\Delta$ from the path metric. In Table 4.2, it is seen that, in the back search of the decoder in steps 1 and 3, the variable θ is set to 1 but then is immediately set to 0 when the decoder attempts a forward move in steps 2 and 4, since the path metric is less than Δ. In step 5, the decoder begins a back search, and the variable θ is set equal to 1 for step 6. However, in decoding

step 6, when the decoder moves forward, the path metric is greater than Δ. Hence, θ is not set to zero and $t\Delta$ cannot be subtracted from the path metric. By decoding step 7, the path metric is now less than Δ, and θ is set equal to zero for step 8. The decoder only back searches when the most probable path segment results in a negative path metric, and as the decoder begins to search forward after the path metric has been increased by Δ, the same most probable path segment must be re-examined. However, when the branch is reached that initiated the back search, the path memory will be between 0 and Δ. Hence, θ is reset equal to zero only when a branch is examined for the first time.

In Table 4.2, the decoded sequence is shown corresponding to the path through the tree diagram. As is seen by the example, in order to decode a message, the sequential decoder may have to back up and change the partially decoded sequence. Therefore, the decoded bits cannot be immediately released to the data user. The number of decoded bits that must be retained in order for the decoder to change the partially decoded segment represents a delay of information to the data user. Thus, an important part of this study is to provide techniques to minimize this number of retained decoded bits.

In Section 2.0, it was pointed out that paths in the tree diagram include remergers. If the sequential decoder follows a path other than the transmitted path which remerges with the transmitted path, then

the sequential decoder will make undetected errors corresponding to the erroneous decoded bits along the unremerged portion of the path. However, it has been shown by theoretical bounds and by simulations that the probability of undetected error decreases exponentially with the memory length K, approximately as $2^{-K/2}$. Therefore, the probability of undetected error can be made arbitrarily small since the sequential decoder does not require a finite memory length code. However, as will be seen in the next section, the probability of error per bit of a sequential decoder is severely dependent on the computational capability (number of branches that may be searched for each received branch) of the decoder. While the probability of error per bit is still completely limited by the complexity of the decoder, under certain conditions, the sequential decoder can achieve lower probability of error per bit for a given complexity than any other decoding technique devised to date.

4.2 PERFORMANCE PARAMETERS

A functional block diagram of a practical sequential decoder is presented in Figure 4.2. The sequential decoder receives quantized code symbols from the demodulator. The forward buffer is used to store incoming data until the decoder can examine the data. Under normal conditions, the forward buffer is relatively empty. However, during periods of a large number of errors from the demodulator (or symbols with low reliability in the quantized case), the decoder must

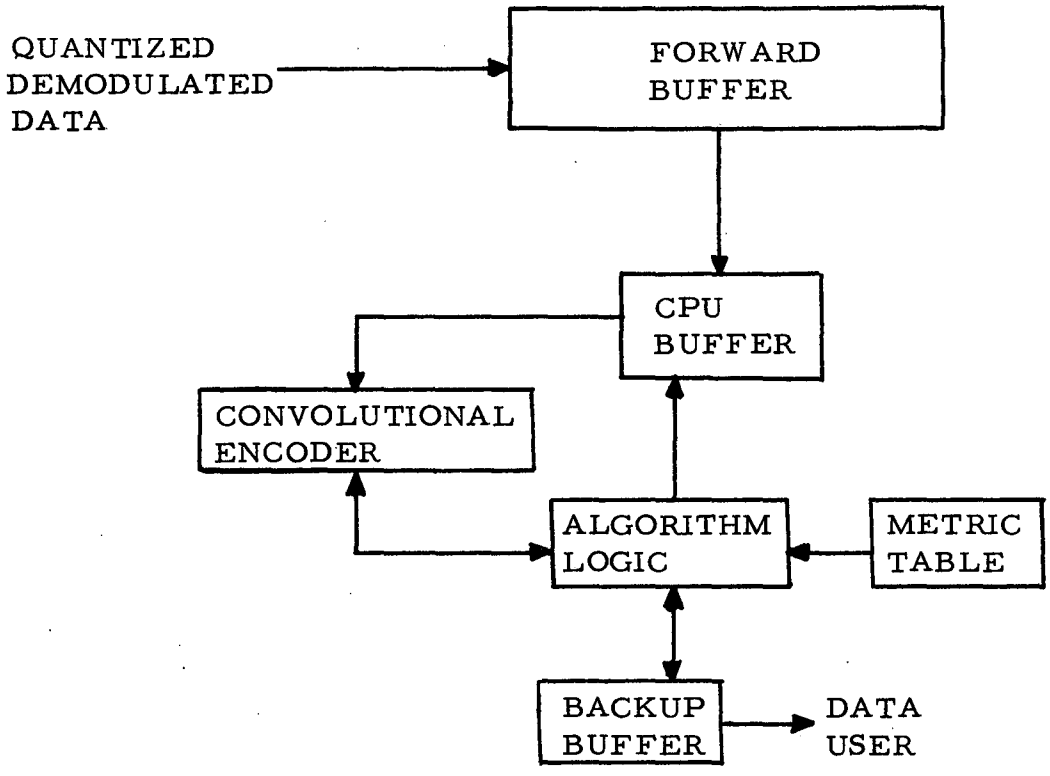


Figure 4.2 Functional Block Diagram of a Practical Sequential Decoder

search many branches in order to find the most probable code word that was transmitted. Thus, during periods of high computational demand on the decoder, the forward buffer stores the incoming data until there is a period of a small number of errors from the demodulator.

The CPU buffer in Figure 4.2 allows the decoder algorithm logic to read several branches at a time from the forward buffer. The backup buffer is used to store the quantized symbols of b received branches corresponding to b past hypothesized bits of the transmitted path. Each time the decoder investigates a received branch, the algorithm logic loads the convolutional encoder with the previous hypotheses to compute the symbols on the particular branch in the code tree, and the branch metric is obtained from the metric table. If the decoder searches forward and extends the path in the tree by one branch, a new received branch is obtained from the forward buffer (or from the CPU buffer). When the new hypothesis and received branch are stored in the backup buffer, one hypothesized bit (the oldest in the backup buffer) is released to the data user.

Computer simulations are necessary to measure the performance of sequential decoding with various design parameters since analytic results are not exact enough for engineering design purposes when the best performance is to be obtained. The computer simulation program of the sequential decoder was written prior to this study by Axiomatix to measure the computations distribution, the backsearch distribution,

and the probability of undetected errors. The program is oriented to the ideal coherent PSK modulation with a white Gaussian noise channel. However, very little modification is necessary to simulate the performance of the sequential decoder in conjunction with other types of modulation and channels. The program is very flexible in terms of the design parameters that may be specified for the sequential decoder. Either rate 1/2 or 1/3 codes can be used up to a maximum memory length of the encoder of 60 stages. The generated samples of the channel can be quantized by Q = 1, 2, or 3 bits. The metric table is generated by the computer program calculating the probability of the quantization assignments as discussed in the previous section. These probabilities of the quantization assignment also could be input data when specifying the metric table. The bias U is also an input parameter. Another input design parameter is how many bits are used to represent each entry in the metric table. The scale factor c is computed such that the most negative entry (which is also always the largest in magnitude) in the table uses all of the available bits for its representation. The remaining design parameter to be specified for the algorithm is the threshold spacing Δ. The value of Δ is specified relative to the unscaled metrics. However, the threshold spacing used by the decoder in the simulation is $c\Delta$ or a scaled value.

To establish the performance of the sequential decoder with various design parameters, statistics are taken on the number of computations (moves) for the decoder to find a path that it can extend one branch

farther into the code tree than was previously reached. Each time a path is extended one branch farther into the code tree, a decoded bit is released to the data user. Thus, these statistics on the computations lead to the computations distribution to decode a bit. This computations distribution to decode a bit is used in the design of a practical sequential decoder to determine the size of the forward buffer and the speed advantage of the logic unit.

Another design parameter for a practical sequential decoder is the size of the backup buffer which determines the decoding delay. To determine the required size of the backup buffer, the backsearch distribution is measured. Each time a path is found by the decoder that may be extended one branch farther into the code tree, the maximum number of nodes is recorded for which the decoder was required to back up to find this new path or to add Δ to the metric so that the present path could be extended. Thus, from the backsearch statistic for each decoded bit, the probability of a given backsearch required by the decoder is determined.

4.2.1 Probability of Undetected Error

To obtain undetected error statistics of the sequential decoder, as each bit is decoded, the bit is compared with the known transmitted bit to test for errors. The measured probability of undetected error for a signal energy per bit/noise density (E_b/N_o) of 4.8 dB, using a systematic code of rate 1/2 with hard decisions on the received

binary symbols, is given in Table 4.3. The algorithm design parameters for this case are those for the optimized rate 1/2 hard decision sequential decoder presented in example, i.e., a branch metric equal to 1 for a double agreement between the received symbols and the symbols on the branch, -5 for a single agreement, -11 for a double disagreement, and threshold spacing $\Delta = 9$. Table 4.3 illustrates that, for memory length larger than 32, the probability of undetected error is negligible for a design of the forward buffer and backup buffer size to obtain an output probability of error equal to 10^{-4} which is used for the voice channels of manned spacecraft missions.

Table 4.3 Sequential Decoder Undetected Error Probability

Memory Length	Probability of Undetected Error
9	4.0×10^{-3}
20	2.8×10^{-4}
32	2.0×10^{-5}

4.2.2 Computation Distribution

The probability $P(C \geq C_m)$, the computations distribution, is the probability that the number of branches C (not necessarily distinct) to be searched for a newly received branch is greater than or equal to some number of branches C_m. An alternate definition is the probability that the number of computations C to decode a bit is greater than or equal to some number of computations C_m. The computations

distribution has been upper bounded by Savage[6] for random codes and by Huth[7] for fixed codes and has been lower bounded by Jacobs and Berlekamp[8] for random codes. These theoretical bounds and computer simulations have indicated that, for C_m much greater than 1 (i.e., $C_m \geq 100$), the computations distribution is Pareto in form or:

$$P(C \geq C_m) \simeq A_c C_m^{-\alpha} \tag{4.3}$$

where α and A_c are given in Table 4.4 for various code rates and values of E_b/N_o that have been measured by computer simulation. The branch metrics for the rate 1/2 hard decision decoder have previously been optimized to only require 5 bits for their representation (i.e., 1, -5, -11, as has been discussed). For this case, the threshold spacing has been optimized, resulting in a scaled value of 9. In the other cases, the threshold spacing is unscaled, as discussed previously. The remaining simulation results presented in Table 4.4 have not been optimized to use the minimum number of bits to represent the branch metrics. However, this will be presented in Section 4.2.4. For comparison and to obtain simple measures of the complexity, a larger number of bits is used to represent the branch metrics than may be eventually needed, and the bias U is chosen to equal the rate R. Also, to guarantee that the probabilty of undetected error is a negligible contribution to the total probability of error, larger code memory lengths are used than may be eventually needed. The computations distribution, the backup distribution, and the complexity of the decoder

Table 4.4 Measured Computations Distribution Parameters

Code Rate R	Memory Length K	Branch Metric Quantization (bits)	Received Symbol Quantization Q (bits)	E_b/N_o (dB)	Threshold Spacing Δ	A_c	α
1/2	32	5	1	4.6	9*	0.212	0.895
				4.9	9*	0.304	1.07
				5.2	9*	0.328	1.187
				5.5	9*	0.389	1.41
	44	12	3	2.5	1.5	0.480	0.907
					3.0	0.309	0.879
					4.5	0.291	0.868
					6.0	0.302	0.848
				3.0	3.0	0.415	1.167
				3.5	3.0	0.775	1.49
1/3	23	12	1	3.5	3.0	0.387	0.723
					4.5	0.394	0.716
				4.0	3.0	0.526	0.954
				4.5	3.0	0.580	1.145
				5.0	3.0	0.624	1.36
			3	2.0	3.0	0.540	0.924
					4.5	0.469	0.887
				2.5	3.0	0.639	1.138
				3.0	3.0	0.696	1.364

*The threshold spacing Δ for the optimized rate 1/2 hard decision is the scaled value while all other threshold spacing values are unscaled.

are all very insensitive to the code memory length in the range 32 to 64. Therefore, it is not an uncommon design practice to use a larger code memory length than may be needed.

While Table 4.4 presents the parameters of the computations distribution that make complexity of the forward buffer a straightforward calculation, it is difficult to obtain an intuitive feeling for the relationship between the choice of algorithm parameters and their resulting performance. Therefore, the measured computations distributions from the computer simulations are presented in Figures 4.3 through 4.6. Figure 4.3 presents the computations distribution for the optimized rate 1/2 hard decision decoder. As E_b/N_o is increased, it may be observed that significant improvements in the computations distribution are obtained. In Figure 4.4, the performance for the rate 1/2 code and three-bit quantization on the received symbols is presented. At E_b/N_o = 2.5 dB, several values of threshold spacing are presented. The best choice for the unscaled threshold spacing Δ is about 3.0. The value of Δ = 4.5 is only slightly worse so that, actually, a fairly wide choice for Δ is possible without significant degradation. Again, a significant improvement in the computations distribution is obtained for larger values of E_b/N_o. Figure 4.5 presents the rate 1/3 hard decision decoder. The choice of Δ = 3.0 still gives a slight improvement over Δ = 4.5, and increasing E_b/N_o gives significant improvement. Finally, Figure 4.6 presents compu-

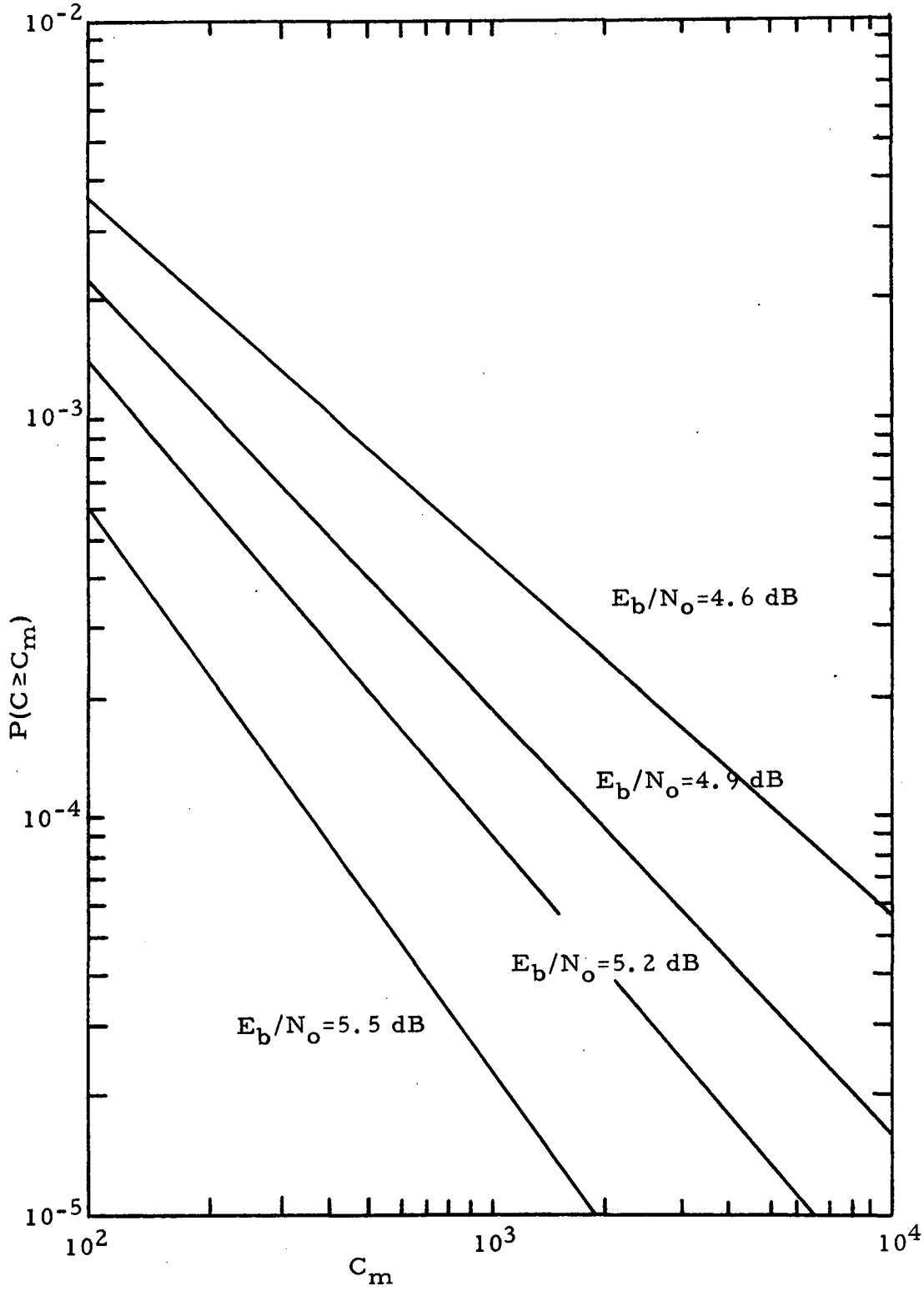


Figure 4.3 Computations Distribution for Rate one-half Hard Decision Sequential Decoder with K=32

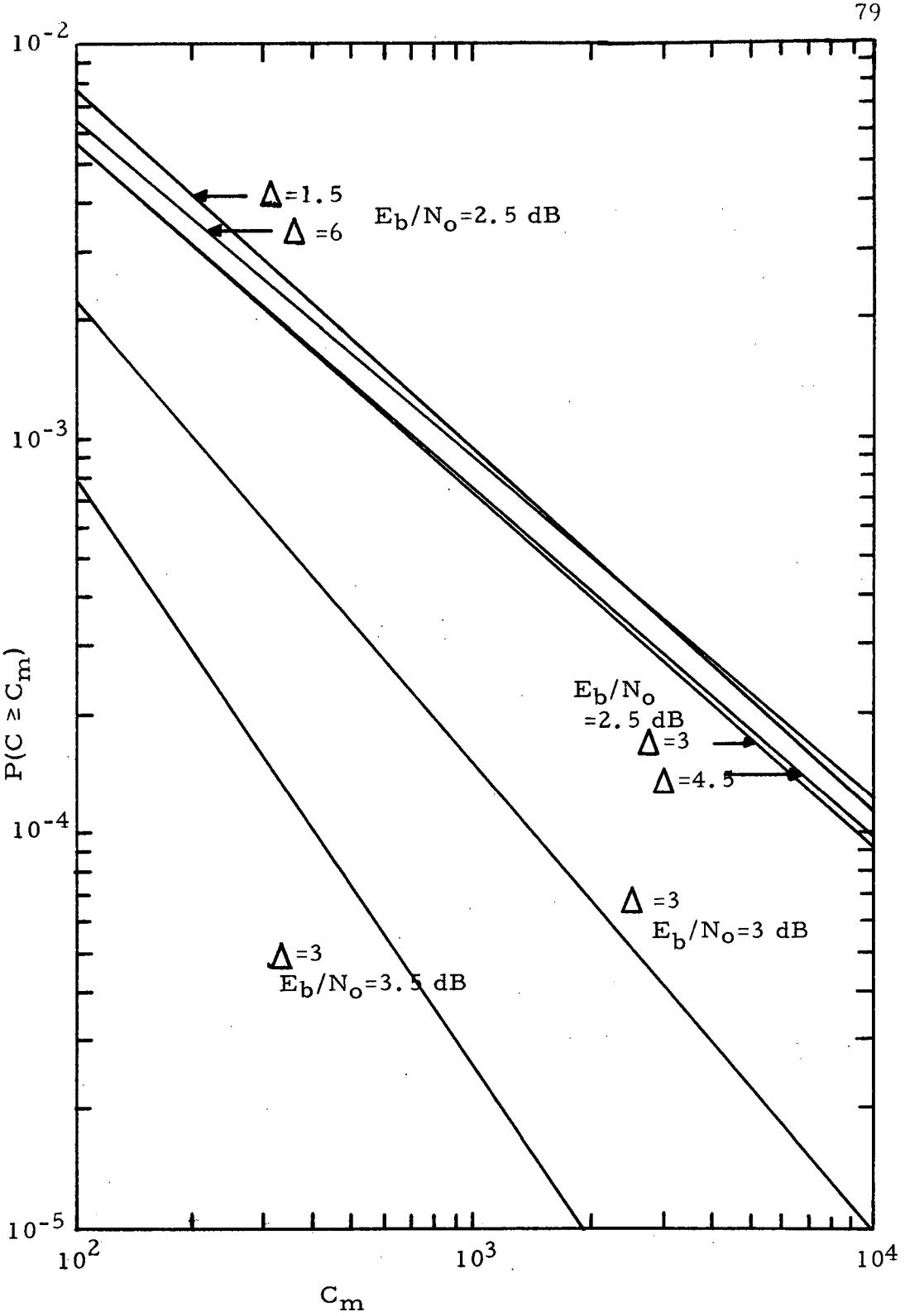


Figure 4.4 Computations Distribution for Rate one-half Three Bit Quantization Sequential Decoder with K=44

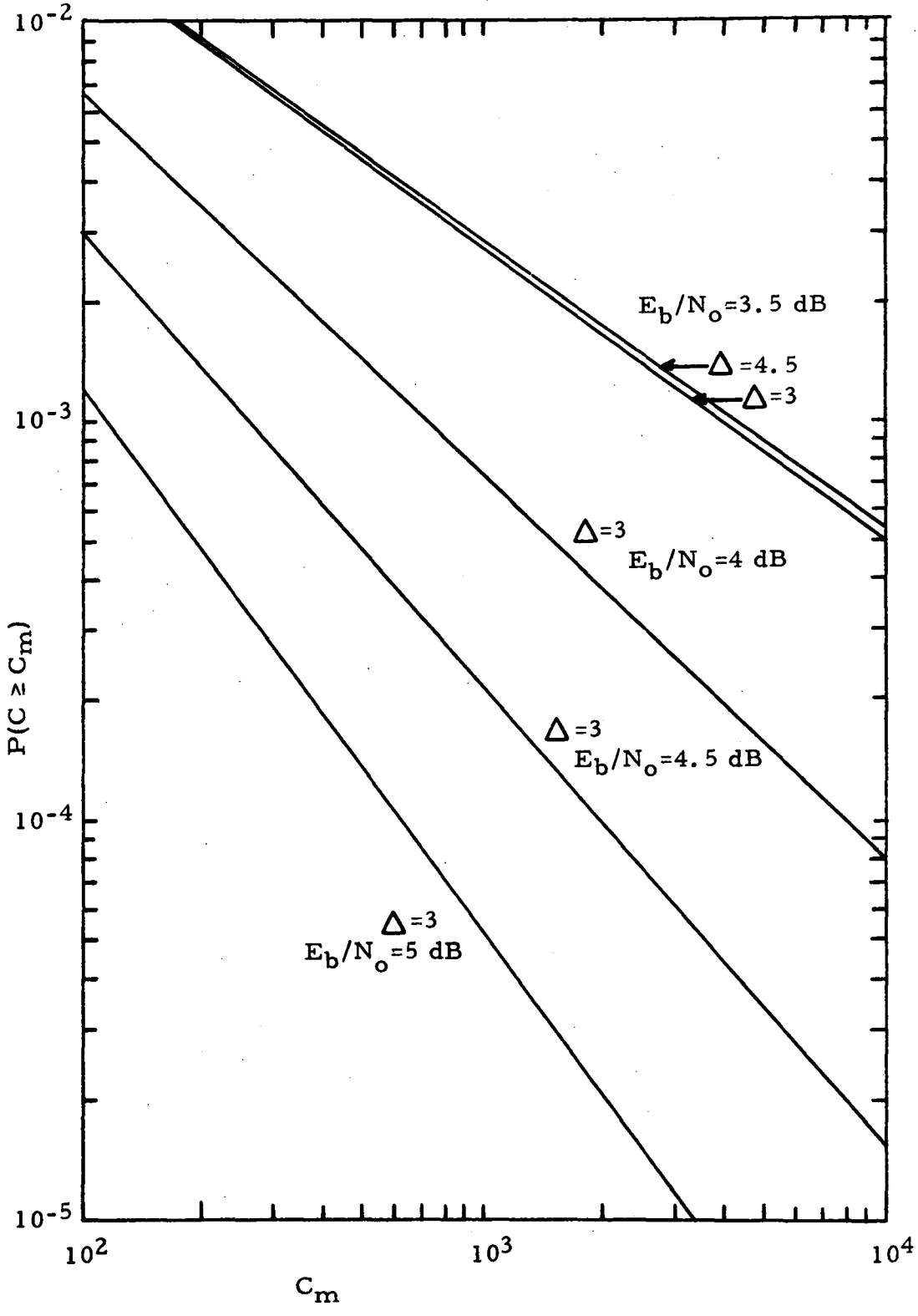


Figure 4.5 Computations Distribution for Rate one-third Hard Decision Sequential Decoder with K=23

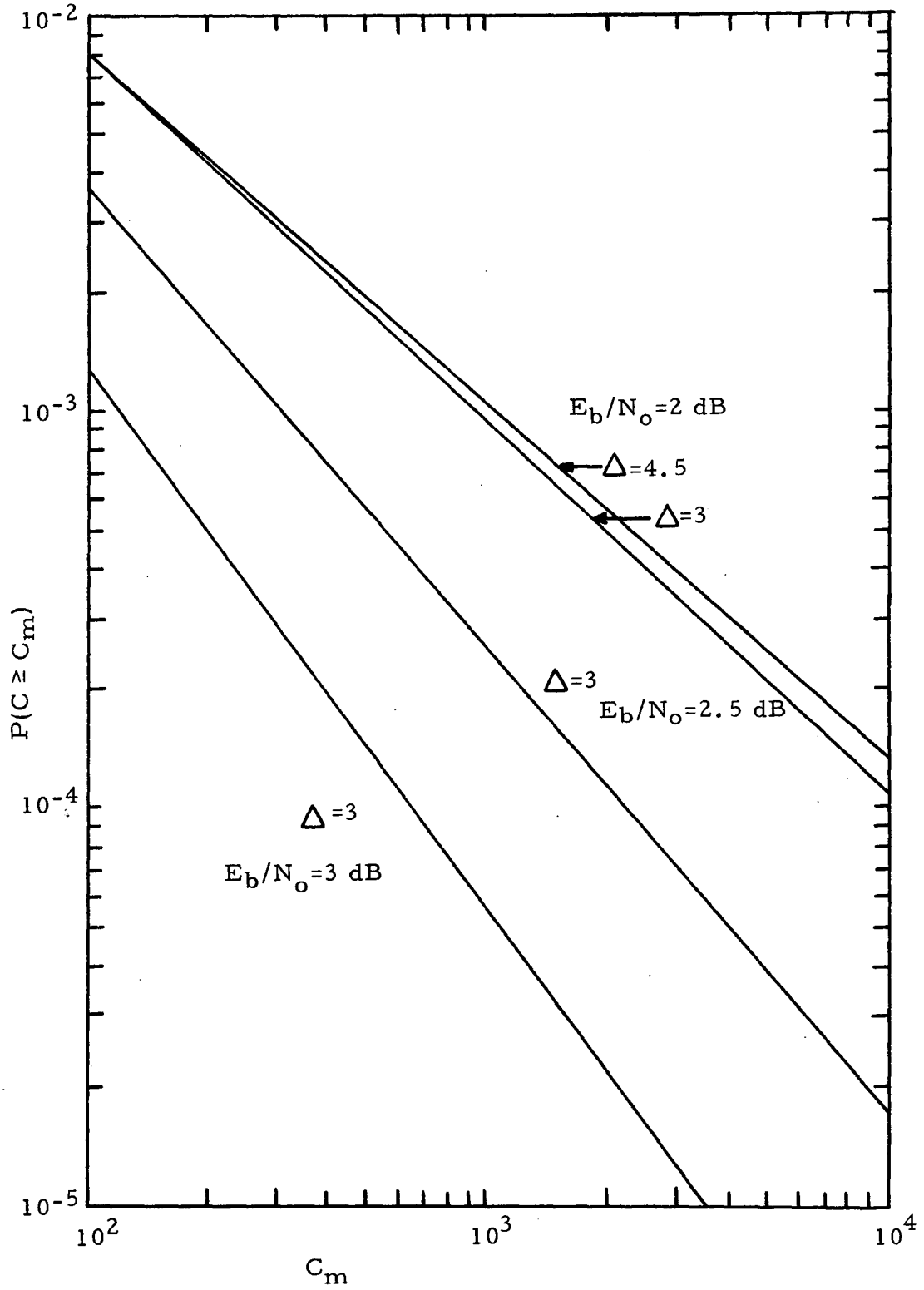


Figure 4.6 Computations Distribution for Rate one-third Three Bit Quantization Sequential Decoder with K=23

tations distribution of the decoder with rate 1/3 and three-bit quanitzation on the received symbols. In this case, the threshold spacing is more critical, and $\Delta = 3.0$ gives much better performance than $\Delta = 4.5$. Comparing the four figures, it is seen that approximately 2.0 dB in E_b/N_o is gained by using three-bit quantization instead of hard decisions on the received symbols, and not quite 0.5 dB in E_b/N_o is gained by using a code rate of 1/3 instead of 1/2.

4.2.3 Backsearch Distribution

The probability $P(b_s \geq b_m)$, the backsearch distribution, is the probability that the number of branches b_s, for which the decoder must back up in its search to find the best path containing a newly received branch, is greater than or equal to some number of branches b_m. The backsearch distribution has been upper bounded by Huth[7] for fixed codes. This theoretical bound and computer simulations have indicated that, for b_m greater than two memory lengths for rate 1/2 and one memory length for rate 1/3, the backsearch distribution is of the form:

$$P(b_s \geq b_m) \simeq A_b 2^{-\beta b_m} \qquad (4.4)$$

where β and A_b are given in Table 4.5 for various code rates and values of E_b/N_o that have been measured by computer simulation. In Section 4.2.4, the number of bits to represent the branch metrics will be optimized in terms of the computations distribution and the backsearch distribution to minimize any degradation that might occur when using fewer bits. However, for initial comparisons of the backsearch distribution, 12 bits were used to represent the branch metrics.

Table 4.5 Measured Backsearch Distribution Parameters

Code Rate R	Memory Length K	Branch Metric Quantization (bits)	Received Symbol Quantization Q (bits)	E_b/N_o (dB)	Threshold Spacing Δ	A_b x 10^{-2}	β x 10^{-2}
1/2	32	5	1	4.6	9*	1.28	6.53
				4.9	9*	0.770	7.05
				5.2	9*	0.270	7.15
				5.5	9*	0.381	9.97
	44	12	3	2.5	1.5	1.24	4.65
					3.0	0.910	4.99
					4.5	0.630	5.45
				3.0	3.0	0.342	5.36
				3.5	3.0	0.139	6.14
1/3	23	12	1	3.5	3.0	1.28	5.90
					4.5	6.54	7.08
				4.0	3.0	4.84	10.2
				4.5	3.0	5.84	13.9
				5.0	3.0	5.11	17.6
	23	12	3	2.0	3.0	2.59	7.66
					4.5	4.94	9.20
				2.5	3.0	6.54	13.2
				3.0	3.0	6.12	17.0

*The threshold spacing for the optimized hard decision rate 1/2 decoder is the scaled value while all other values of threshold spacing are unscaled.

Table 4.5 presents the parameters of the backsearch distribution that simplify the calculation of the backup buffer complexity. However, it is difficult to make comparisons between the choice of algorithm parameters using their resulting performance. Therefore, the measured backsearch distributions from the computer simulations are presented in Figures 4.7 through 4.10. Figure 4.7 presents the backsearch distribution for the optimized rate 1/2 hard decision decoder. As E_b/N_o is increased, it may be observed that significant improvements in the backsearch distribution are obtained. In Figure 4.8, the performance for the rate 1/2 code with three-bit quantization on the received symbols is presented. At E_b/N_o = 2.5 dB, the performance for several values of threshold spacing is presented. As the threshold spacing is increased, the backsearch distribution is improved until $\Delta = 4.5$, where there is no improvement obtained by increasing the threshold spacing to $\Delta = 6.0$. However, the best choice for the threshold spacing in terms of the computations distribution was $\Delta = 3.0$. Therefore, there is a trade-off between the computations distribution and the backsearch distribution in choosing the threshold spacing. Except when the speed advantage is extremely large, as it is for the data rates of the voice channels, the value of threshold spacing is used that gives the best computations distribution, since the forward buffer represents the majority of the complexity and decoding delay. Therefore, most of the computer simulations used $\Delta = 3.0$ as the value of threshold spacing.

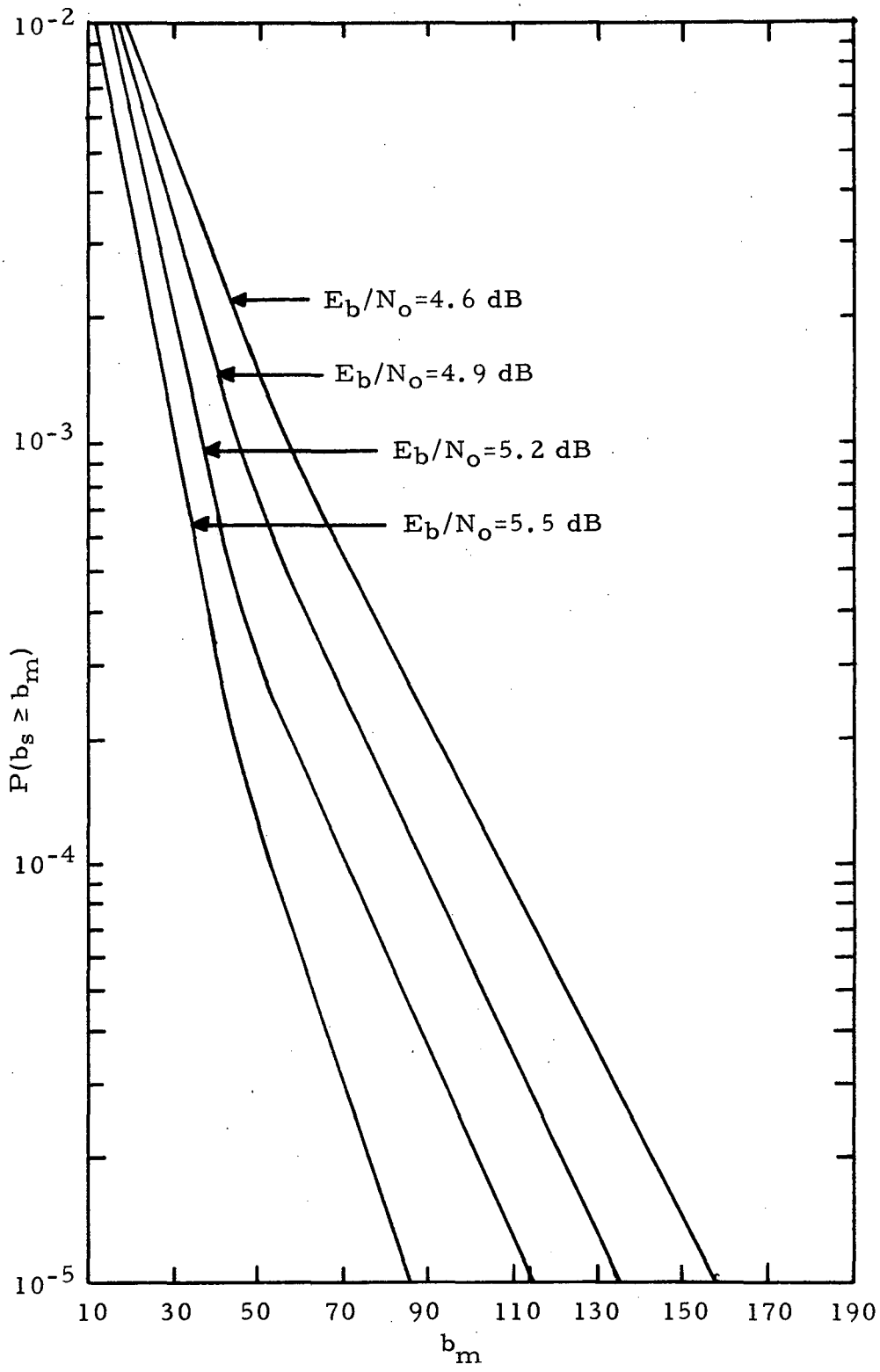


Figure 4.7 Backsearch Distribution for Rate one-half Hard Decision Sequential Decoder with K=32

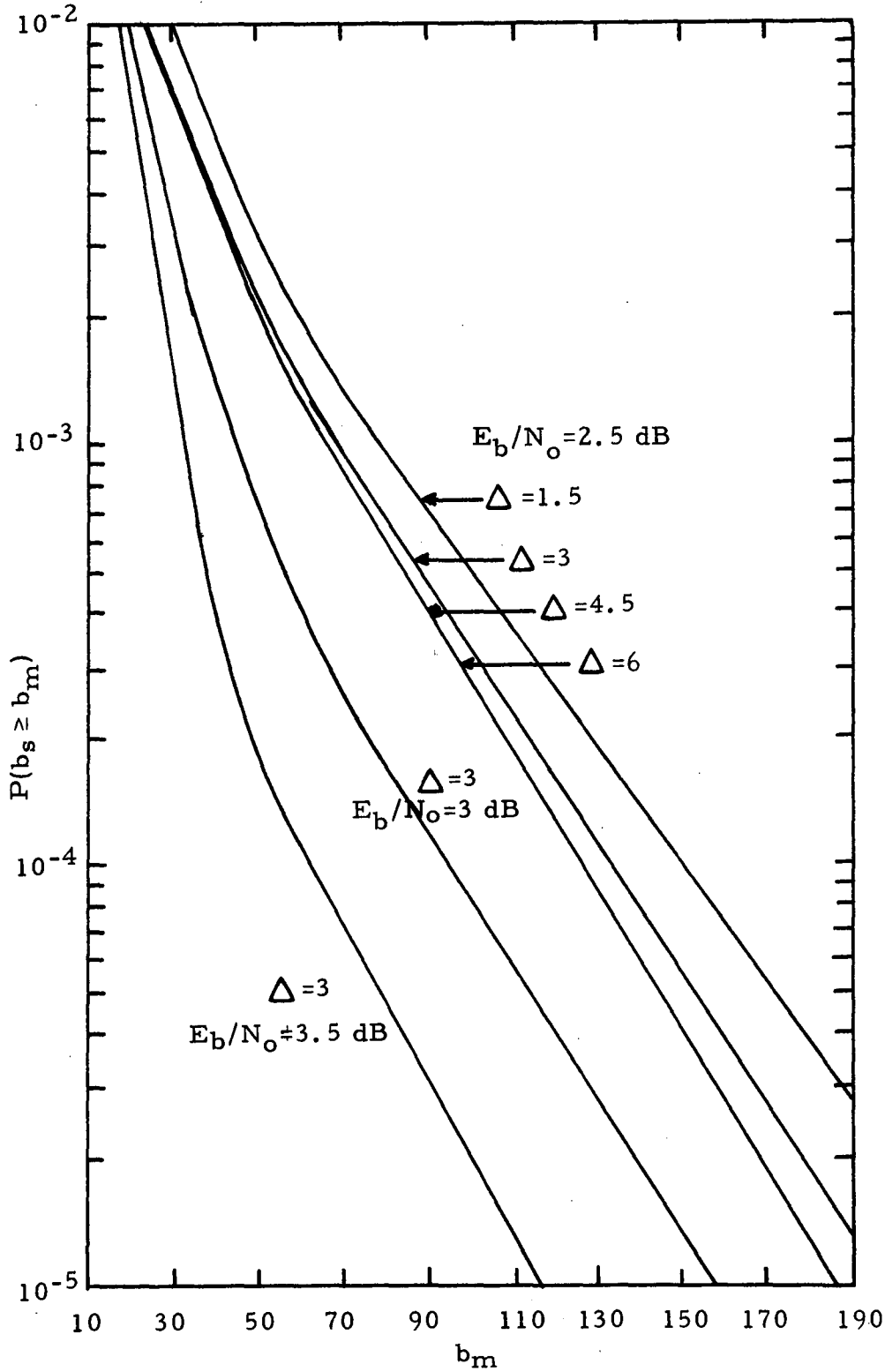


Figure 4.8 Backsearch Distribution for Rate one-half Three Bit Quantization Sequential Decoder with K=44

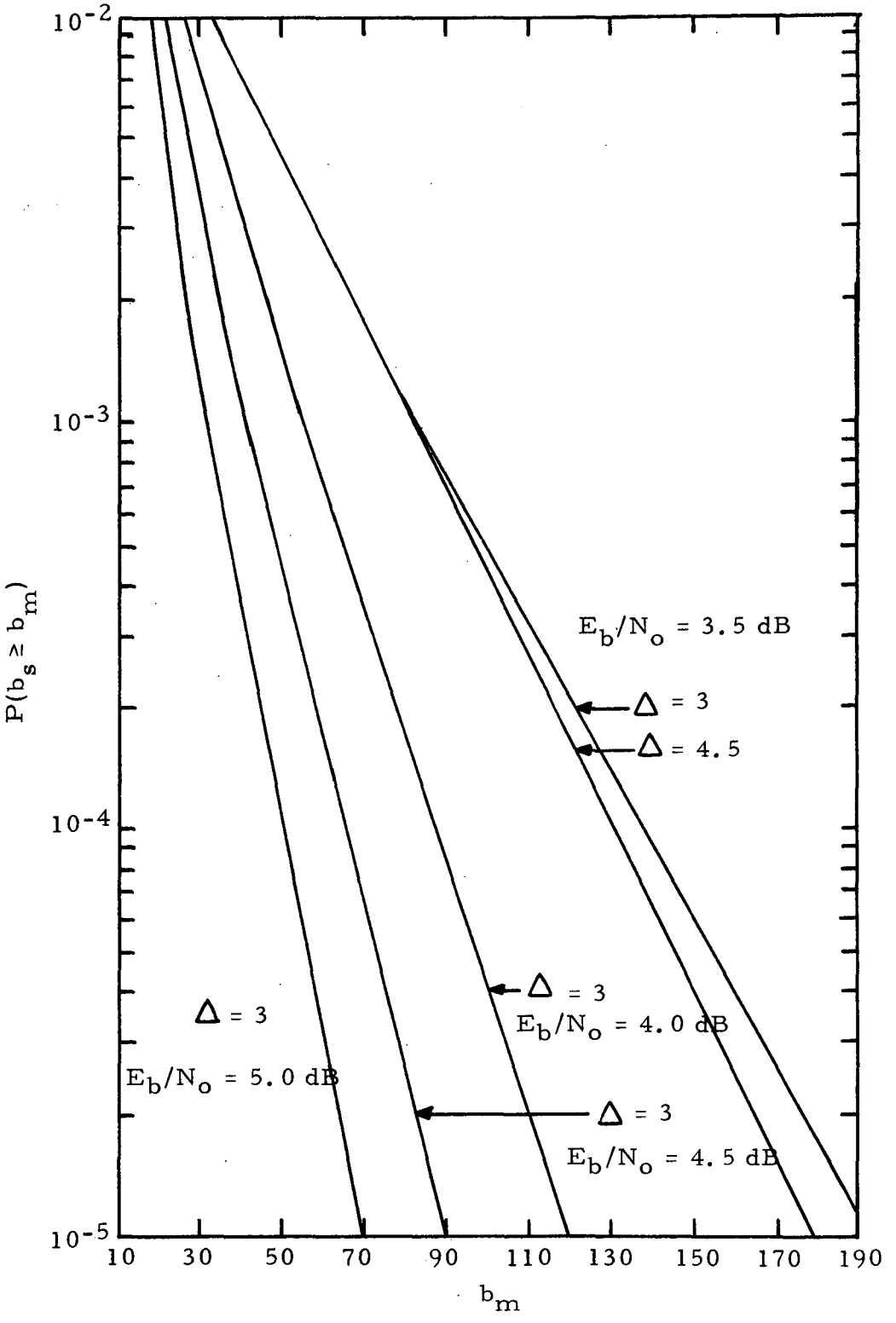


Figure 4.9 Backsearch Distribution for Rate 1/3 Hard Decision Sequential Decoder with K = 23

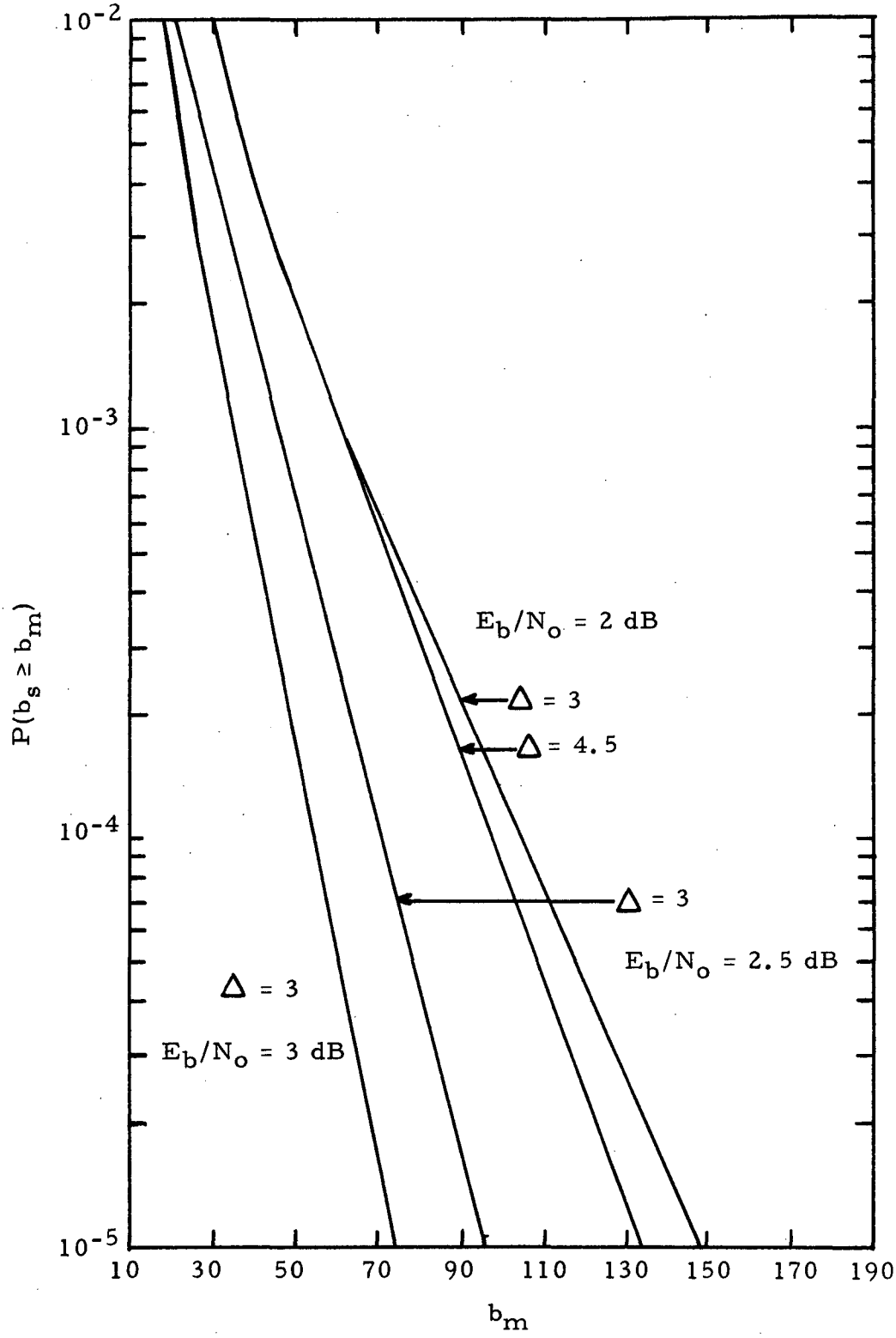


Figure 4.10 Backsearch Distribution for Rate 1/3 Three Bit Quantization Sequential Decoder with K = 23

Comparing the backsearch distributions for rate 1/3 in Figures 4.9 and 4.10 with the backsearch distributions for rate 1/2, it may be observed that the slopes of the distribution for rate 1/3 are much steeper than for rate 1/2. Thus, fewer branches must be stored for backup with code rate 1/3. The size of the backup buffer does not depend on the speed advantage of the decoder and, hence, not on the data rate. For each branch to be stored in the backup buffer, Q/R+1 storage bits are needed.

4.2.4 Degradation Due to Metric Table Size

To determine the computations distribution and backsearch distribution, the number of bits used to represent each branch metric was $N_b = 12$, except for the code rate 1/2, hard decision case, where N_b has been previously optimized to be equal to 5. In this section, the performance versus N_b is presented.

The entries in the metric table have N_b bits to represent the best branch metric, a bit to indicate whether the best branch is the zero branch or the one branch, and N_b bits to represent the difference between the best metric and the other metric. Thus, $2N_b+1$ bits are needed for each entry in the metric table. There are 2^{nQ} possible metric intervals where the n symbols on a branch are quantized to Q bits. Therefore, the metric table has 2^{nQ} words of $2N_b+1$ bits each.

Table 4.6 presents the results of simulations for various values of N_b in terms of the parameters A_c and α of the computations

TABLE 4.6 Performance Versus Branch Metric Quantization

Code Rate R	Received Symbol Quantization Q (bits)	E_b/N_o (dB)	Branch Metric Quantization N_b (bits)	Computation Distribution Parameters		Backsearch Distribution Parameters	
				A_c	α	A_b x10^{-2}	β x10^{-2}
1/2	3	3.0	12	0.415	1.167	0.342	5.36
			10	0.516	1.194	0.930	6.95
			8	0.562	1.205	0.930	6.95
			7	0.530	1.175	0.389	4.94
			6	0.326	0.915	1.0	3.69
1/3	1	4.5	12	0.580	1.145	5.84	13.9
			10	0.670	1.182	8.90	15.6
			8	0.588	1.110	7.50	13.4
			7	0.506	0.973	1.60	6.26
	3	2.5	12	0.639	1.138	6.54	13.2
			10	0.549	1.107	7.50	13.4
			8	0.564	1.116	7.50	13.4
			7	0.584	1.097	5.00	11.6

distribution and A_b and β of the backsearch distribution. It may be seen that, for $R = 1/2$ and $Q = 3$, the performance greatly deteriorates for $N_b = 6$. For $R = 1/3$ and $Q = 1$ or 3, the performance for $N_b = 6$ is so poor that the decoder can no longer decode the data fast enough to make a computer simulation practical. This poor performance occurs since there is not fine enough quantization to specify branch metrics such that the incorrect paths are different enough from the correct path, resulting in the decoder searching too many of the incorrect paths.

4.3 DECODING RESYNCHRONIZATION

For a finite forward buffer and backup buffer, it can be seen from the computations distribution and the backsearch distribution that the forward buffer may fill and overflow and the decoder may attempt a backsearch to the limit of the backup buffer. When either of these two events occur, the decoding process is interrupted and must be resynchronized. The requirement necessary for resynchronization is a memory length K of information bits without error to restart the convolutional encoder used by the decoder on the correct path.

The most familiar resynchronization scheme, and the simplest conceptually, is block resynchronization. In this case, periodically, a memory length of predetermined information bits is used to form the tail of the block which is transmitted. Since any choice of information

bits is allowed for the tail, usually a sequence with good correlation properties is used (i.e., a pseudonoise sequence or a Barker sequence). Thus, acquisition of block synchronization (i.e., the location of the block tail) is performed by searching the received symbols for the known sequence spaced Γ-K branches apart, where Γ is the block length. The tracking of block synchronization is performed by either a delay lock loop or a tau jitter loop, both of which are commonly used in pulsed pseudonoise modulation systems. However, most NASA manned missions already have transmitted synchronization sequences for other subsystems. Therefore, block synchronization and branch synchronization can be derived from the system synchronization and the complexity of acquisition and tracking will not be considered in this study.

The complexity of block resynchronization is in the transmitter where the data must be buffered during insertion of the tail sequence. Since the tail is a memory length of information bits, a buffer the size of the convolutional encoder is needed. However, the symbol rate is not exactly 1/2 or 1/3 of the input data rate with block resynchronization, and this may present problems in some systems. In many of the systems considered, multiplexing is performed, and the symbol rate needed for block resynchronization ceases to be a problem. The remaining disadvantage of block resynchronization is the rate loss due to the tail sequence that transmits no information.

Using the block resynchronization technique, when the decoding process is interrupted, the remaining information bits in the block are output to the data user. The decoder can now be moved to the beginning of the next block. Since a memory length with predetermined information digits was transmitted, the decoder can load its encoder with these known information digits, and resynchronization is achieved. The errors from the interruption of the decoding process occur since the remaining undecoded information bits in the resynchronization block are output with channel errors. Thus, the probability of error due to interruption of the decoding process is given by the probability of decoding interruption multiplied by the average number of errors each time the decoding process is interrupted.

$$P_e = P_I(P_{ue}\,\Gamma_1 + p\,\Gamma_2) + P_{ue} \tag{4.5}$$

where

P_I = probability of decoding interruption

P_{ue} = probability of an undetected error per bit

p = probability of a channel error

Γ_1 = number of decoded bits in a resynchronization block

Γ_2 = number of undecoded bits in the resynchronization block including the number of bits in the backup buffer

Since on the average about one-half the bits in the resynchronization block have been examined before an overflow,

$$\Gamma_1 = (1/2)\Gamma - b$$
$$\Gamma_2 = (1/2)\Gamma + b \tag{4.6}$$

where

Γ = total number of bits in the resynchronization block

b = number of bits in the backup buffer

The probability of undetected error is normally designed to be negligible in comparison with either the probability of decoding interruption or the probability of channel error ($P_{ue}\,\Gamma_1 \ll p\,\Gamma_2$). Thus, equation (4.5) may be rewritten as:

$$P_e = (1/2\,\Gamma + b)P_I p \tag{4.7}$$

The rate for block resynchronization is given by:

$$R' = R\left(\frac{\Gamma - K}{\Gamma}\right) \quad \text{for } \Gamma \geq K \tag{4.8}$$

Another resynchronization scheme that is possible is statistical resynchronization. Using this scheme, there is no rate loss, and the symbol rate may be exactly 1/2 or 1/3 of the input data rate. Also, there is no additional complexity in the transmitter to handle the resynchronization. For systems where transmitted synchronization is not available, extra hardware to perform acquisition and tracking of block synchronization is not required. Thus, statistical resynchronization can have a tremendous advantage in complexity over block resynchronization.

With statistical resynchronization, when a forward buffer overflow occurs, the decoder is moved further into the received data stream, and hard decisions on a memory length of information bits are used to load the decoder's replica convolutional encoder to restart decoding. For hard decision demodulation, a resynchronization strategy is to

establish the point of restart as the "origin" such that the decoder cannot back up past it. The decoder is declared resynchronized when the backup buffer is again filled. As in block resynchronization, channel errors in the information bits are not corrected between buffer overflow and resynchronization. The probability of error over this period is just equal to the channel probability of error. If the computations required to resynchronize are too large, then the choice of the hypothesis bits used to load the decoder's replica encoder is probably incorrect, and the resynchronization attempt should be rejected. A new resynchronization attempt is begun by moving the decoder farther into the received data stream a specified number of branches.

For rate 1/2 and hard decisions made on the demodulated symbols, the probability of choosing a memory length of correct hypothesis bits to load the replica is:

$$P_{rt} = (1 - p)^K \tag{4.9}$$

For channel probability of error p = 0.045 corresponding to E_b/N_o = 4.6 dB and R = 1/2, P_{rt} = 0.229 with memory length K = 32, and repeated guessing would give a correct memory length of bits in about five trials on the average. The decoder is allowed to return to the "origin" H times before it is rejected. Table 4.7 presents the results of simulations to measure the probability of resynchronization, where P_{rch} is the probability that a resynchronization attempt is rejected with the correct hypothesis bits in the replica encoder. It should be noted that

Table 4.7 Measured Probability of Resynchronization

Channel Probability of Error p	Memory Length K	Probability of Resynchronization		H	P_{rch}
		Predicted	Measured		
0.045	32	0.229	0.345	3	4.16×10^{-3}
			0.281	2	1.66×10^{-2}
			0.234	1	0.1
			0.134	0	0.462
0.035	32	0.308	0.454	3	0
			0.382	2	3.02×10^{-3}
			0.338	1	3.92×10^{-2}
			0.222	0	0.365

the measured probability of resynchronization is considerably larger for $H > 1$ than equation (4.9) predicts. The large measured probabilities of resynchronization result from the ability of the decoder to restart even when errors have occurred in the initial hypothesis bits loaded into the replica encoder. The density of the possible errors in the initial hypothesis bits loaded into the replica encoder that can occur in a successful resynchronization attempt is illustrated in Figure 4.11.

To produce the curves in Figure 4.11, each successful resynchronization was tested for one or more errors in a group of five hypothesis bits. For example, if a successful resynchronization had an error in the third and the seventh hypothesis bits, then the group of the 1-5 hypothesis bits and the group of 6-10 hypothesis bits would each be credited with containing errors. The fraction of successful resynchronization attempts having errors in groups of five is illustrated by horizontal

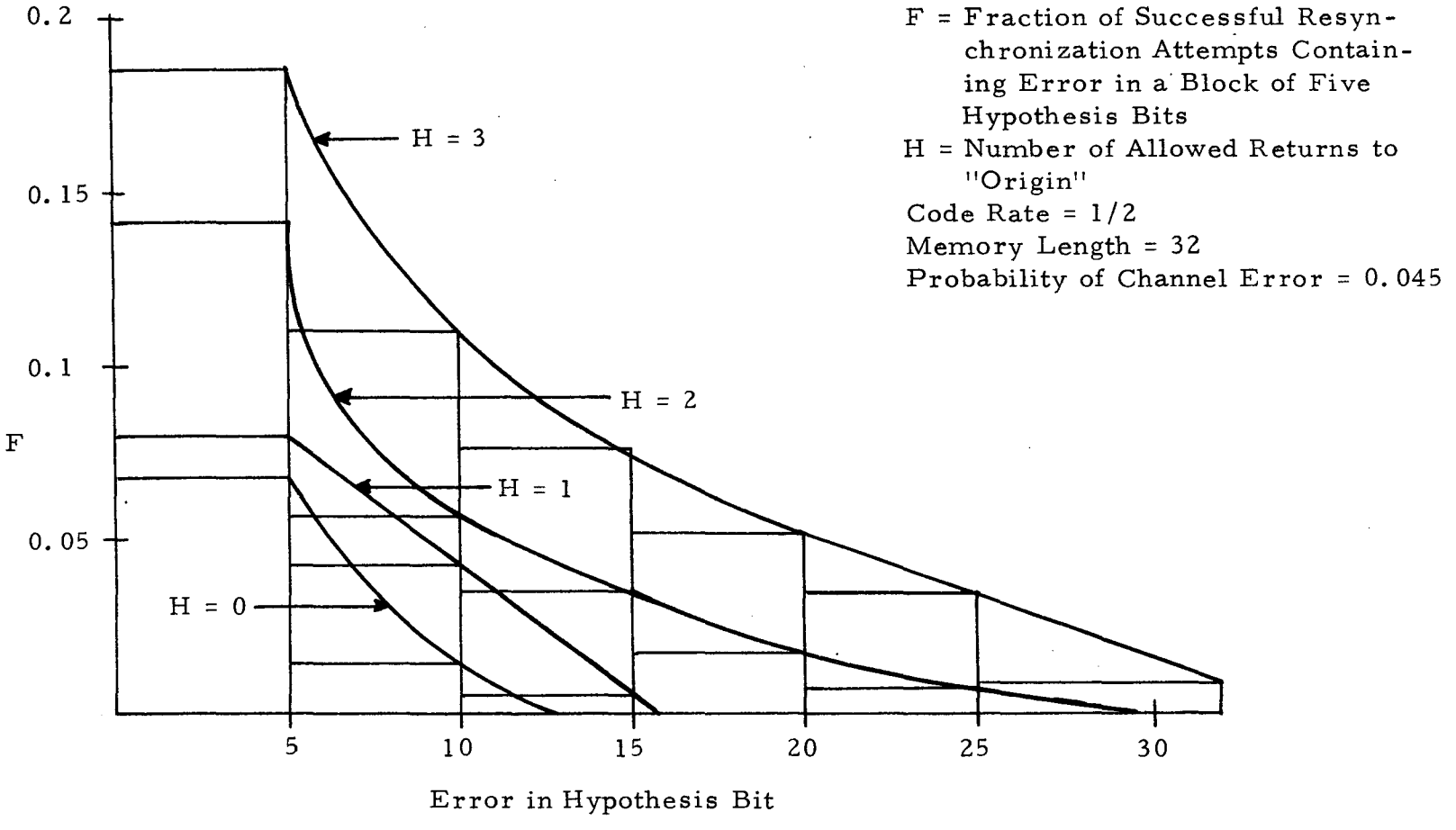


Figure 4.11 Density of Errors in Initial Hypothesis for a Given Resynchronization

lines in Figure 4.11. The curves connecting the horizontal lines are drawn to illustrate the shape of the density. As H decreases, only errors in the beginning of the initial hypothesis bits will allow the decoder to resynchronize. This decrease in allowed positions of errors within the initial hypothesis bits partly accounts for the decrease in the probability of resynchronization as H decreases. The additional decrease in probability of resynchronization as H decreases results from the increase in P_{rch}.

Resynchronization is declared when the decoder has again filled the backup buffer because, at this point, as the decoder searches newly received data, decoded bits must be released to the data user. By simulation, it has been found that the computations distribution for resynchronization is just the distribution to decode b bits if H = 3. However, the computations distribution for a resynchronization attempt that will eventually be unsuccessful has an extremely large required number of computations to advance b bits. Therefore, a strategy is to specify a certain number of allowed computations. If the decoder exceeds the specified number of computations, C_m, the resynchronization attempt is rejected. Resynchronization of the decoder is achieved if:

(1) $H_i \leq H$

(2) $C_s < C_m$

are both true, where:

H_i = the actual returns to the "origin"

H = the allowed returns to the "origin"

C_s = actual computations to accept a resynchronization attempt.

Thus, to derive the probability of restart $P_r(C_m)$, the following development can be used:

$$P_r(C_m) = P(r/H_i \leq H)P(H_i \leq H) \tag{4.10}$$

where $P(H_i \leq H)$ is the measured probability of resynchronization in Table 4.7.

$$P(r/H_i \leq H) = P(C_s < C_m) = 1 - P(C_s \geq C_m) \tag{4.11}$$

where $P(C_s \geq C_m) = bP(C \geq C_m)$ and $P(C \geq C_m)$ is the computations distribution given in Section 4.2.2. Thus,

$$P_r(C_m) = (1 - bP(C \geq C_m))P(H_i \leq H) \tag{4.12}$$

and

$$Y(C_m) = \frac{1}{P_r(C_m)} \tag{4.13}$$

where $Y(C_m)$ is the average number of attempts before the decoder is resynchronized.

Equations (4.12) and (4.13) assume that each resynchronization attempt is independent of all others. To achieve this in a practical design, the decoder must be advanced at least K+1 branches for each resynchronization attempt. However, the decoder must output the information bits in the backup buffer for the first resynchronization attempt to be independent. Hence:

$$s = b + Y(C_m)(K + 1 + g) \quad \text{for } g \geq 0 \tag{4.14}$$

where s is the number of information bits output to the data user with

channel errors uncorrected. Also:

$$C_m = S(J + K + 1 + g) \tag{4.15}$$

where S is the speed advantage of the decoder, which is the number of branches the decoder can examine for each branch received (i.e., $S = R_c/R_d$ where R_c is the computation rate of the algorithm logic unit and R_d is the data rate). The value of J in equation (4.15) is given by:

$$J = \left\lfloor \frac{b}{\lfloor Y(C_m) + 1 \rfloor} \right\rfloor \tag{4.16}$$

Note $\lfloor x \rfloor$ is the smallest integer contained in x.

The decoder resynchronization design procedure is to minimize s with respect to C_m. However, since $Y(C_m)$ is a complex function of C_m, the derivative is not easily obtained, and the equation for C_m contains functions of C_m on each side of the equation. Hence, the solution for the minimum value of s is most easily found by an iterative procedure.

An iterative procedure for finding the minimum value of s is as follows:

1. Let g = 0 and compute $Y(C_m)$ for $C_m = S(K + 1)$.
2. Using the integer value of $(Y(C_m + 1)$, recompute C_m using equation (4.15).
3. Recompute $Y(C_m)$ with the value of C_m from step 2.
4. If the integer value of $(Y(C_m) + 1)$ is the same as the value of step 2, compute s. Otherwise, using the new integer value $(Y(C_m) + 1)$, go to step 2 and repeat the procedure.

5. Using J computed in step 4 (equation (4.16) as part of the computation of s), compute C_m for g = 1, C_m = S(J + K + 2). Compute $Y(C_m)$ using this value of C_m. If the integer value of $Y(C_m)$ + 1 is not the same as the value in step 4, go to step 2 with g = 1.

6. Compute s for g = 1. If s is larger than the value obtained in step 4, stop and use the parameters corresponding to g = 0. Otherwise, minimize a with respect to C_m = S(K + 1 + g) for integer values of g. Compute J using the new integer value of $(Y(C_m) + 1)$, and the design value g_d will be g - J.

The statistical resynchronization strategy for three-bit quantization on the demodulated symbols is to make hard decisions on the information bits for loading the decoder's replica convolutional encoder while the remaining quantization is used as a reliability measure on the bits. An information bit is said to be questionable if the reliability quantization bits are 00 (i.e., the two quantization levels closest to zero). If there are more than N_E questionable information bits in the bits to be loaded in the replica encoder, then the decoder is moved one position further into the received data stream to test another K information bits to load the replica encoder. The value of N_E is chosen to minimize the number of information bits output to the data user with channel errors uncorrected. When acceptable K information bits are found,

the decoder is restarted with this point as the "origin." The decoder is allowed a given number of computations to decode b (backup buffer size) information bits. If the decoder exceeds the allowed number of computations, then one of the most questionable information bits is complemented and the decoder is again restarted. When all combinations of the questionable bits are tried and the decoder has again exceeded the allowed number of computations, the decoder is moved further into the received data stream to test another K information bits to load the replica encoder. The decoder is declared resynchronized if b information bits are decoded within the allowed number of computations.

The probability of resynchronization if only hard decision information is used is given by:

$$P_{rh} = (1 - bP(C \geq C_m))(1-p)^K \tag{4.17}$$

If only N_E questionable bits are allowed for a resynchronization attempt then the probability of resynchronization for a selected K information bits is given by:

$$P_{rs} = (1 - p_0)^{K-N_E}(1-bP(C \geq C_m/2^{N_E})) \tag{4.18}$$

The value of p_0 is the probability of error with high reliability (i.e., an error in one of the remaining three highest quantization levels). The probability of selecting an acceptable set of K information bits is:

$$P_N = \sum_{i=0}^{N_E} \binom{K}{i} p_1^i (1-p_1)^{K-i} \tag{4.19}$$

The value p_1 is the probability of an information bit having low reliability.

The decoder is moved b branches further into the data stream before the first resynchronization attempt. At this point, the set of K information bits is tested. If this set is not acceptable, then the decoder is moved one more branch into the data stream. On the average, $L = \lceil 1/P_N \rceil$ branches are examined before a set of information bits is loaded into the decoder's replica convolutional encoder. Therefore, the allowed number of computations, C_m, for the first resynchronization attempt is:

$$C_m = S(b + L) \tag{4.20}$$

where S is the speed advantage of the decoder. After the first attempt, the decoder is advanced at least K+1 branches before a set of K information bits is tested for the next attempt. Thus, the number of computations allowed for the second and following attempts is:

$$C_m = S(K + 1 + g + L) \tag{4.21}$$

where the value of g is used to optimize the number of allowed computations. The number of information bits output, on the average, to the data user with the channel errors uncorrected is:

$$I = b + L + (K + 1 + g + L)(1 - P'_{rs})/P_{rs} \tag{4.22}$$

The value of P'_{rs} is the probability of resynchronization on the first attempt.

A comparison of block and statistical resynchronization for complexity and performance is contained in the following sections, where the parameters of the sequential decoder are defined.

4.4 COMPLEXITY OF THE SEQUENTIAL DECODER

The sequential decoder algorithm was presented in Section 4.1. To obtain the complexity of the algorithm logic, a detailed algorithm logic flow diagram is presented in Figure 4.12. In this section, the complexity of each block in the flow diagram will be found to give an overall complexity of the decoder. The flow diagram in Figure 4.12 is for the TTL logic. However, for the high data rate (9.1 Mbps), a large forward buffer is needed. Several manufacturers presently have available large random-access memories on cards with a read/write cycle time of 400 nanoseconds. Using the MECL III algorithm logic unit for this 9.1 Mbps data rate, 110 million branches/second is to be processed. Therefore, the CPU buffer must store 44 branches in high speed memory. In this case, each time the decoder reaches a new branch, the CPU buffer address counter is incremented. When the CPU buffer is empty, the address counter requests another 44 branches to be read from the forward buffer. Otherwise, the MECL III algorithm logic has the same flow as the TTL logic.

The data received from the demodulator is stored until all n quantized symbols for a rate 1/n code are received. Thus, nQ bits of storage are necessary to collect the quantized data to be stored in the forward buffer. When all n symbols have been received, the data clock increments the forward buffer input address counter. A branch of data is read out of the forward buffer when the decoder moves forward and

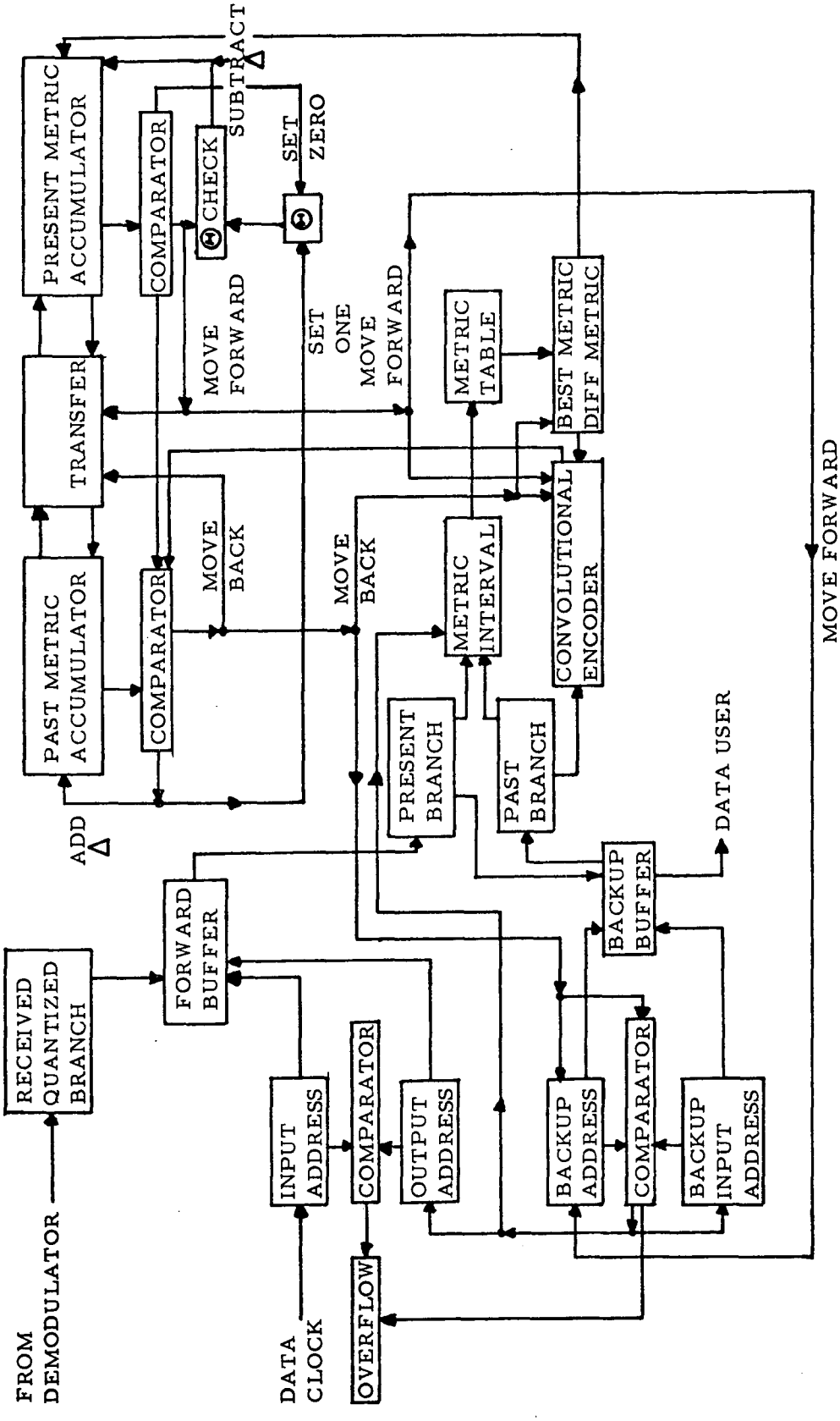


Figure 4.12 Detailed Sequential Decoder Algorithm Logic Flow Diagram

increments the output address counter. The data to be read into the forward buffer has the highest priority so that no data is lost. A comparator is hard-wired between the two address counters to detect when the buffer is full or empty. The complexity associated with the inputting of data is:

$$\text{INPUT} = nQ(B+1) + 3\lceil \log_2 B \rceil \qquad (4.23)$$

for TTL where B is the forward buffer size in branches. For MECL III, the complexity associated with the input is:

$$\text{INPUT} = nQ(B+44) + 3\lceil \log_2 B \rceil + 7(nQ(44)+6) \qquad (4.24)$$

where the last term represents the CPU buffer and addressing. The constant 7 is the relative weight of the MECL III cost including parts, design, and packaging with respect to TTL. This weighting is a subjective judgment on the part of the author and should not be considered exact. The relative weighting will change with the changing cost of parts, new MSI announcements, and experience using logic families.

The backup address counter receives the move forward and the move back signals to count up or down. A comparator is used to detect when the backup address counter equals the backup input address counter. If the two counters are equal and a move back signal is present, then the backup buffer overflows. Otherwise, if the counters are equal, a signal is generated by the comparator to read a new branch from the forward buffer and a new branch into the backup buffer. As a new branch is read into the backup buffer, the hypothesis bit in the input address

location is read out to become the decoded bit to the data user. The complexity of the backup portion of the decoder is:

$$\text{BACKUP} = (nQ+1)b + 3\lceil \log_2 b \rceil \qquad (4.25)$$

where b is the backup buffer size in branches. For the MECL III decoder, the complexity in equation (4.25) is multiplied by seven.

To generate the branch metric, a present branch and a past branch are stored. If the signal for the new branch is generated from the backup comparator, then the present branch is used to compute the metric interval; otherwise, the past branch is used. The convolutional encoder generates the branches along the path. To compute the metric interval, the convolutional encoder shifts in a zero. If the decoder moves along the branch, then the bit corresponding to the best metric is inserted into the convolutional encoder in place of the zero used initially. The metric interval is computed by "exclusive-OR"-ing the sign bits of the appropriate quantized branch with the output of the convolutional encoder for an input zero. The metric interval is used as the address to the metric table producing as its output the hypothesis bit for the best branch, the best branch metric, and the difference metric, requiring a total of $2N_b+1$ bits for each entry in the metric table. The best metric is transferred to the present metric accumulator to be added. If the decoder backed up and then moves forward along the worse branch, then the difference metric is added to the present metric accumulator (the difference metric is actually negative so the effect is to obtain

the branch metric for the worse branch). If, as the decoder backs up, the hypothesis bit does not correspond to the best branch metric, then a signal is sent to the past metric accumulator comparator to check the accumulator value for the direction of the next move. The complexity of determining the hypothesis bits and the branch metrics for the accumulator is:

$$\text{HYPOTHESIS} = n(3Q+1+K) + (2^{nQ}+1)(2N_b+1) \qquad (4.26)$$

The first term represents the branch storage, the metric interval computation and the convolutional encoder. The convolutional encoder is implemented as the convolutional impulse response with a K stage shift register for the past hypothesis bits. While this implementation is slightly more complex than usual, it has the advantage of only one gate delay and is used for high speed computation. The second term in equation (4.26) is the metric table and storage for the output branch metric.

The metric accumulators must be large enough to allow the decoder to move through a very noisy segment of the received data. To obtain a bound on this quantity, let W_m equal the branch metric corresponding to n symbol metrics with the metric interval 011. This is the most positive metric. Therefore, if there were no noise on the channel, then the decoder could back to the end of the backup buffer and return to the front with, at most, a metric of $W_m b$. Thus, the metric accumulator need not be larger than $\lceil \log_2 W_m b \rceil$ plus a sign bit. The past metric comparator only detects the sign bit of the accumulator. The present

metric comparator detects the sign bit and if the accumulator is less than Δ. Hence, the complexity of the present metric comparator is $\lceil \log_2 W_m b \rceil + 1$. As the decoder moves forward and backward, the past metric is parallel transferred to the present metric and vice versa. The direction the decoder moves is determined by the values of the accumulators. If the present metric accumulator becomes negative, the past metric accumulator is compared. If both metrics are negative, then Δ is added to the past metric and θ is set to one. If only the present metric is negative, then a move back signal is generated. If the present metric is positive, then a move forward signal is generated. When the present metric is greater than or equal to Δ and $\theta = 0$, then Δ is subtracted from the present metric. The complexity of metric accumulators, transfers, and comparators is:

$$\text{ACCUMULATOR} = 5(\lceil \log_2 W_m b \rceil + 1) + 4 \qquad (4.27)$$

The overall complexity of the sequential decoder is given by:

$$\begin{aligned}\text{DECODER} = {} & nQ(B+1) + 3\lceil \log_2 B \rceil + (nQ+1)b + 3\lceil \log_2 b \rceil \\ & + n(3Q+1+K) + (2^{nQ}+1)(2N_b+1) \\ & + 5(\lceil \log_2 W_m b \rceil + 1) + 4 \qquad (4.28)\end{aligned}$$

using TTL. For MECL III, the complexity is:

$$\begin{aligned}\text{DECODER} = {} & nQ(B+44) + 3\lceil \log_2 B \rceil + 7nQ(44)+6+(nQ+1)b \\ & + 3\lceil \log_2 b \rceil + n(3Q+1+K) + (2^{nQ}+1)(2N_b+1) \\ & + 5(\lceil \log_2 W_m b \rceil + 1) + 4 \qquad (4.29)\end{aligned}$$

These equations do not include the external complexity required for branch synchronization or resynchronization after a decoding interruption.

4.4.1 Buffer Complexity

The size of the forward buffer and the backup buffer can be determined from the computations distribution and the backsearch distribution, respectively. The computations distribution determines the size of the forward buffer by determining the probability of forward buffer overflow, P_o. It has been found that P_o can be approximated by:

$$P_o \simeq A_c(SB)^{-\alpha} \tag{4.30}$$

where B is the number of branches that can be stored in the forward buffer and S is the speed advantage. Equation (4.30) is a good approximation for $\alpha \geq 1$, $S \geq 10$, and $B < 10^6$. For $\alpha < 1$, the speed advantage must be larger than the average number of computations, which grows exponentially as α is decreased below one. The computation of the algorithm logic is 12.5, 25, and 100 million computations per second using TTL, MECL II, and MECL III logic families, respectively.

The size of the backup buffer in branches is given by:

$$b = \frac{1}{\beta} \log_2(A_b/P_s) \tag{4.31}$$

where P_s is the probability of the decoder reaching the end of the backup buffer due to a backsearch. The parameters A_c, α, A_b and β are given in Tables 4.4, 4.5 and 4.6.

A reasonable design value for both P_o and P_s is 10^{-6} if the desired probability of error per bit is to be 10^{-4}. Using these design values, Table 4.8 presents the size of the metric table, the forward buffer, and the backup buffer for various data rates and values of N_b. The number of storage bits in the forward buffer is $n\Omega B$ and the number of storage

bits in the backup buffer is $(nQ+1)b$. Also, the parameters for the minimum implementation complexity is indicated for various data rates. For example, with $R = 1/2$ and $Q = 3$, the minimum implementation complexity corresponds to $N_b = 8$ for all data rates. However, for $R = 1/3$ and $Q = 3$, the minimum implementation complexity corresponds to $N_b = 7$ for the 19.2 kbps and the 38.4 kbps data rates and to $N_b = 8$ for the remaining data rates.

4.4.2 Decoding Resynchronization Complexity

The block and resynchronization schemes are evaluated with respect to performance and complexity in this section. Using equations (4.7) and (4.8) in conjunction with size of the forward buffer and backup buffers given in Table 4.8, the performance of block resynchronization can be determined. The probability of decoding interruption P_I is equal to the sum of P_o and P_s since this sum is much less than one.

By optimizing with respect to block length Γ, it is found that, for $R = 1/2$, the best value of Γ is 1024 branches, which is better than 512 or 2048 branches by 0.1 dB in E_b/N_o. For $R = 1/3$, $\Gamma = 1024$ branches still gives the best performance, but $\Gamma = 512$ branches gives as good a performance while $\Gamma = 2048$ branches is about 0.1 dB in E_b/N_o worse. Figure 4.13 presents the performance using $\Gamma = 1024$ branches for $R = 1/2$ and $1/3$, $Q = 1$ and 3 bit quantization. At output probability of error equal to 10^{-4}, Table 4.9 summarizes the performance.

Statistical resynchronization provides significant performance improvement over the block resynchronization performance summarized

TABLE 4.8 Buffer Complexity Versus Branch Metric Quantization

* Minimum Implementation

Code Rate R	Received Symbol Quantization Q (bits)	E_b/N_o (dB)	Branch Metric Quantization N_b (bits)	Metric Table Size (storage bits)	Backup Buffer Size (storage bits)	Forward Buffer Size (storage bits)				
						Data Rate				
						19.2 kbps	38.4 kbps	57.6 kbps	76.8 kbps	9.1 Mbps ($x10^4$)
1/2	1	4.9	5	44	546	396	792	1,188	1,584	2.34
	3	3.0	12	1,600	1,533	612	1,224	1,836	2,448	3.62
			10	1,344	1,351	552	1,104	1,656	2,208	3.28
			8	1,088	1,351	546*	1,092*	1,638*	2,184*	3.24*
			7	960	1,694	678	1,356	2,034	2,712	4.01
			6	232	2,520	11,520	23,040	34,560	46,080	68.4
1/3	1	4.5	12	200	456	576	1,152	1,728*	2,304*	3.40*
			10	168	420	609*	1,218*	1,827	2,436	3.63
			8	136	480	690	1,380	2,070	2,760	3.86
			7	120	888	3,363	6,726	10,089	13,452	19.98
	3	2.5	12	12,800	1,230	1,773	3,546	5,319	7,092	10.49
			10	10,752	1,080	2,160	4,320	6,480	8,640	12.78
			8	8,704	1,080	1,782	3,564	5,346*	7,128*	10,53*
			7	7,680	1,220	2,142*	4,284*	6,426	8,568	12.69

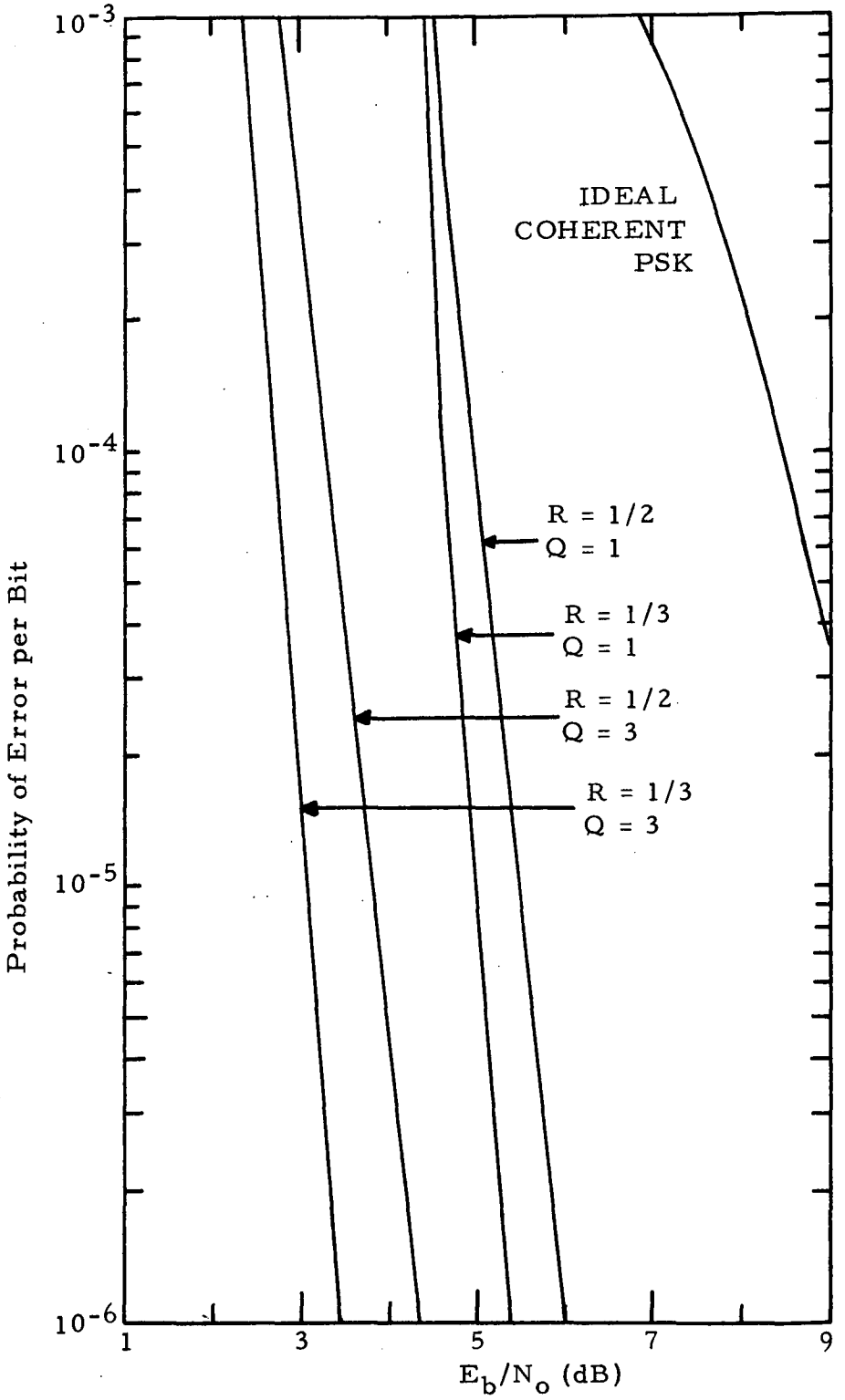


Figure 4.13 Sequential Decoder Performance Using Block Resynchronization with Code Rate R and Q Bits of Quantization

Table 4.9 Performance of Sequential Decoding Using Block Resynchronization at $P_e = 10^{-4}$

Code Rate R	Received Symbol Quantization Q (bits)	Required E_b/N_o (dB)	Coding Gain over Ideal Coherent PSK (dB)
1/2	1	4.95	3.45
	3	3.2	5.2
1/3	1	4.6	3.8
	3	2.7	5.7

in Table 4.9. By using the iterative design procedure presented in Section 4.3, it is found that the minimum value of s is 280 for E_b/N_o = 4.6 dB with K = 32 and b = 182 branches, while the minimum value of s is 255 for E_b/N_o = 5.2 dB. Since the speed advantages obtained, even using TTL with four-voice channels, are so large, the minimum value of s does not decrease as the speed advantage is increased.

For comparison of the two resynchronization techniques, Figure 4.14 presents the performance of each technique for the rate 1/2 hard decision decoder with K = 32, b = 182, B = 198, S = 651, and the block size Γ = 1024 branches, which gives the best performance for block resynchronization. It is seen that statistical resynchronization has 0.38 dB better performance than block resynchronization at an output probability of error per bit of 10^{-4}. Thus, the rate 1/2 hard decision decoder with statistical resynchronization has a coding gain of 3.8 dB over the uncoded ideal coherent PSK.

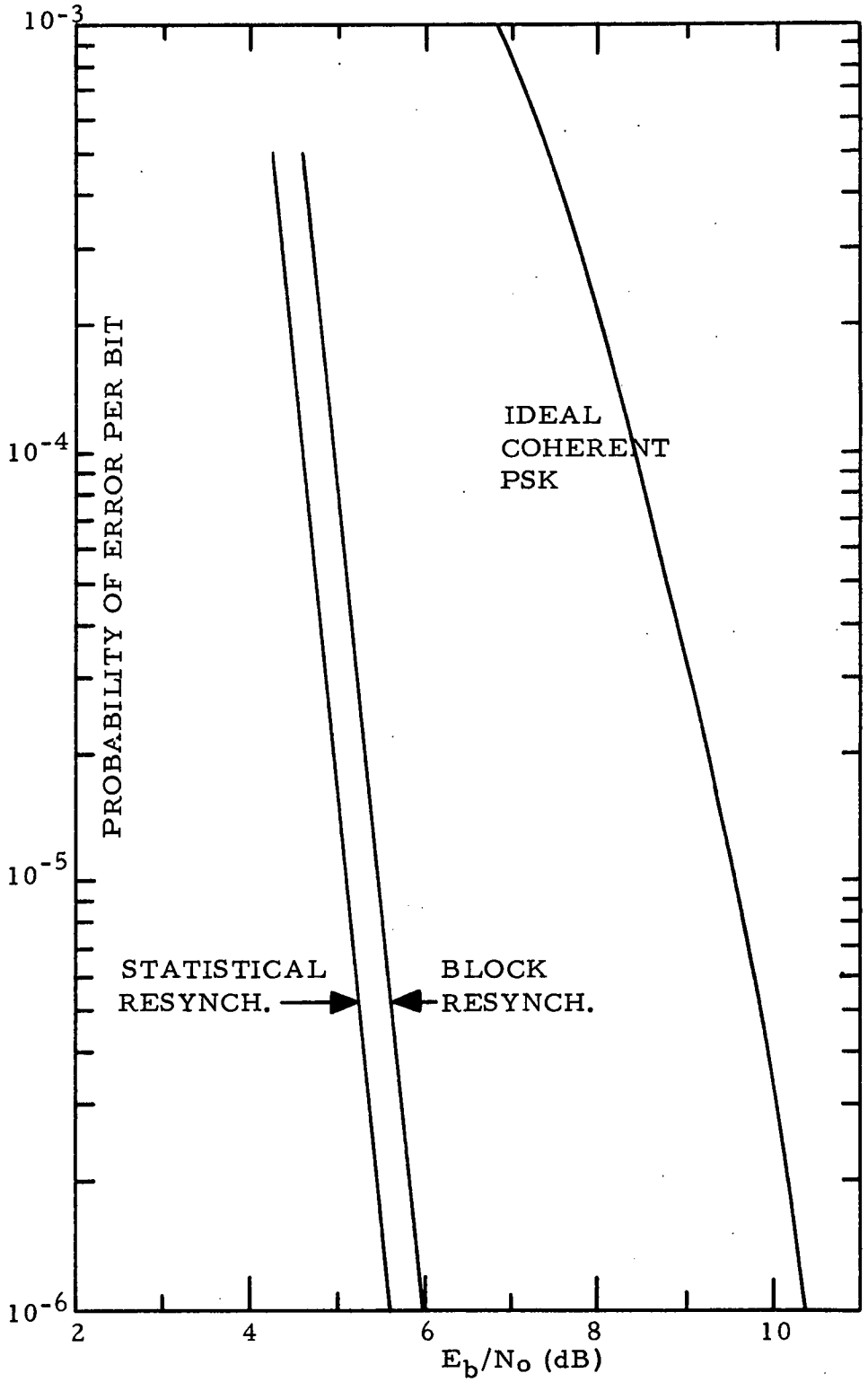


Figure 4.14 Comparison of the Performance of Sequential Decoding Using Block Resynchronization and Statistical Resynchronization

For rate 1/3 and hard decisions on the demodulated symbols, the probability of resynchronization, P_{rt}, can somewhat pessimistically be expressed as $P_{rt} = (1-p)^K$ where p is the channel probability of error per symbol. For K = 23 and E_b/N_o = 4, 4.5, and 5 dB, using coherent PSK modulation, the probability P_{rt} is equal to 0.093, 0.126, and 0.218, respectively. Again by using the iterative design procedure, the number of information bits output to the data user with the channel errors uncorrected is, on the average, 374, 305, and 225 for E_b/N_o = 4, 4.5, and 5 dB, respectively. This design assumes a backup buffer of 114 branches.

As an example of the performance of statistical resynchronization with three-bit quantization, consider the 19.2 kbps data rate using TTL giving a speed advantage of 651. For rate 1/2 with memory length K = 44 and b = 193 at E_b/N_o = 3.0 dB, the design that minimizes I is N_E = 4 and g = 0, giving I = 306. If only hard decision information is used, the equivalent I equals 1423, compared with 705 using block resynchronization. For rate 1/3 with memory length K = 23 and b = 122 at E_b/N_o = 2.5 dB, the design that minimizes I is N_E = 3 and g = 0, resulting in I = 191. The equivalent value of I, using only hard decision information, is 870 compared with 634 using block resynchronization.

Table 4.10 summarizes the performance of statistical resynchronization at output probability of error per bit of 10^{-4}. The improvement by using statistical resynchronization over block resynchronization is 0.4 dB for code rate 1/2 and 0.2 dB for code rate 1/3 with hard

decisions demodulation and 0.3 dB with three-bit quantization.

Table 4.10 Performance of Sequential Decoding Using Statistical Resynchronization at $P_e = 10^{-4}$

Code Rate R	Symbol Quantization Q (bits)	Required E_b/N_o (dB)	Coding Gain over Ideal Coherent PSK (dB)
1/2	1	4.55	3.85
	3	2.8	5.6
1/3	1	4.4	4.0
	3	2.4	6.0

To implement statistical resynchronization, an alternate technique can be used to handle the decoder backing up to the end of the backup buffer. In a majority of the cases, when the sequential decoder searches to the end of the backup buffer, the decoder is following the transmitted path. Therefore, treating the end of the backup buffer as the origin of the code tree, the path metric is increased by Δ and the decoder searches forward. However, if the decoder is not following the transmitted path when the end of the backup buffer is reached, then treating the end of the backup buffer as the origin will result in either an undetected error or, more likely, a forward buffer overflow. Using this technique, it can be assumed that the probability of decoding interruption P_I is the same as before (i.e., $P_I = P_o + P_s$) but all decoding interruptions will occur as forward buffer overflows. The sequential decoder using this technique

can be designed with an output buffer of B bits. The output buffer provides the interface between the decoder and the data user so that the data user may receive data at a constant rate. The output buffer is full if the forward buffer is empty, and the output buffer is empty if the forward buffer is full. Therefore, if the forward buffer is full, then a decoding interruption occurs and b undecoded information bits are shifted out of the backup buffer into the empty output buffer. Note that the sequential decoder design should be for more branches in the forward buffer than in the backup buffer if this technique is used.

As the b bits are shifted into the output buffer undecoded, b branches are shifted into the backup buffer from the forward buffer. For the hard decision case, the last K bits shifted into the output buffer are also shifted into the convolutional encoder to restart the decoding process. Resynchronization is declared when the decoder reaches the front of the backup buffer and a new branch is requested from the forward buffer. If K+1 branches are received before resynchronization is declared, then K+1 bits are shifted undecoded out of the backup buffer into the output buffer with the last K bits also being shifted into the convolutional encoder to restart the decoder for another resynchronization attempt. As the K+1 bits are shifted out of the backup buffer, K+1 branches are shifted from the forward buffer into the backup buffer. Thus, the complexity for the hard decision case is one bit to record

overflow, one bit to record resynchronization, and a gate so that, if resynchronization is in process, an overflow can only reject the resynchronization attempt. In addition, the B bit output buffer shift register and a counter of $\lceil \log_2 K+1 \rceil$ bits to indicate that K+1 bits have been received are needed. Thus, the total hard decision statistical resynchronization complexity is:

$$\text{Resynchronization} = B + \lceil \log_2 K+1 \rceil + 3 \qquad (4.32)$$

For the three-bit quantization case, as the b bits are shifted into the output buffer, the last K quantized information bits are shifted into an examination buffer. In this examination buffer, the reliability bits are examined for 00. If there are N_E or less information bits with 00 reliability bits, then the information bits are loaded into the convolutional encoder and the decoding is restarted. If there are more than N_E information bits with 00 reliability bits, then a quantized information bit is shifted from the backup buffer into the examination buffer. Also, an undecoded bit is shifted from the backup buffer to the output buffer and a branch is shifted from the forward buffer to the backup buffer. After a resynchronization attempt has begun, if the decoder returns three times to the end of the backup buffer, then the state of an N_E bit counter is exclusive-ORed with the bits of 00 reliability if the counter has not exceeded its allowed count. The allowed count of the N_E bit counter is 2^N, where N is the actual number of information bits with 00 reliability. If K+1

branches are received before resynchronization is declared, then K+1 bits are shifted undecoded out of the backup buffer into the output buffer with the last K quantized information bits into the examination buffer. The complexity for the three-bit quantization is then the complexity for the hard decision case plus the examination buffer and logic. Thus, the total three-bit quantization statistical resynchronization complexity is:

$$\text{Resynchronization} = B + \lceil \log_2(K+1) \rceil + 6 + 3(K+N_E) \qquad (4.33)$$

Block resynchronization also requires the B bit output buffer.

The beginning of the blocks may have to be tracked and initially required if this is not incorporated into the system synchronization. In addition, the data in the transmitter must be buffered during insertion of the tail sequence. Thus, because of the relative simplicity of statistical resynchronization and its performance improvement over block resynchronization, only statistical resynchronization will be considered in the complexity of the sequential decoder.

4.5 PERFORMANCE VERSUS COMPLEXITY

The complexity of the sequential decoder with statistical resynchronization has been specified by equations in the previous section. Using these equations and the forward buffer size, the backup buffer size and the metric table size given in Table 4.8, the complexity can be calculated. Table 4.11 presents a summary of the sequential decoder complexity with statistical resynchronization with the required

TABLE 4.11 Overall Complexity of Sequential Decoder for Various Data Rates at Output Probability of Error per Bit of 10^{-4}

Code Rate R	Received Symbol Quantization Q (bits)	E_b/N_o (dB)	Data Rate kbps	Sequential Decoder Complexity (complexity bits)
1/2	1	4.55	19.2	1,364
			38.4	1,961
			57.6	2,558
			76.8	3,122
			9.1×10^3	40,933
	3	3.8	19.2	3,528
			38.4	4,077
			57.6	4,706
			76.8	5,343
			9.1×10^3	58,391
1/3	1	4.4	19.2	1,633
			38.4	2,448
			57.6	3,209
			76.8	3,977
			9.1×10^3	52,526
	3	2.4	19.2	11,589
			38.4	13,981
			57.6	16,058
			76.8	18,038
			9.1×10^3	190,268

E_b/N_o to obtain an output probability of error per bit of 10^{-4} in conjunction with the ideal coherent PSK channel. The complexity for rate 1/3 and Q = 1 is somewhat greater than rate 1/2 and Q = 1 with only 0.15 dB improvement in the required E_b/N_o. There is an 0.4 dB gain of the rate 1/3 over the rate 1/2 with Q = 3, but the complexity is about four times larger for the rate 1/3.

SECTION 5.0

VITERBI AND SEQUENTIAL DECODER PERFORMANCE VERSUS COMPLEXITY

In the preceding sections, the performance for the Viterbi and sequential decoders was presented from simulation results. The complexity of these decoders was presented in terms of complexity bits by equations so that many other data rates and system parameters may be determined for future systems. From careful analysis of the various portions of each decoder, the number of integrated circuit chips can be determined for some logic family and hence the cost. As an example, the complexity of 9.1 Mbps data rate decoder with R = 1/2, K = 3, and Q = 3 is 1,378 complexity bits and requires 180 TTL integrated circuit chips. Alternately, a 9.1 Mbps data rate decoder with R = 1/2, K = 8, and Q = 3 has a complexity of 33,818 complexity bits and requires about 5500 TTL integrated circuit chips.

Using the complexity measure described in this report, a performance versus complexity comparison can be made for the Viterbi and sequential decoders. Figures 5.1 and 5.2 present the comparison for R = 1/2 and 1/3, respectively. For rate 1/2 and hard decisions on the demodulated symbols and the same complexity, the sequential decoder requires 1.3 dB less E_b/N_o than the Viterbi decoder at the 19.2 kbps data rate and an output probability of error per bit of 10^{-4}.

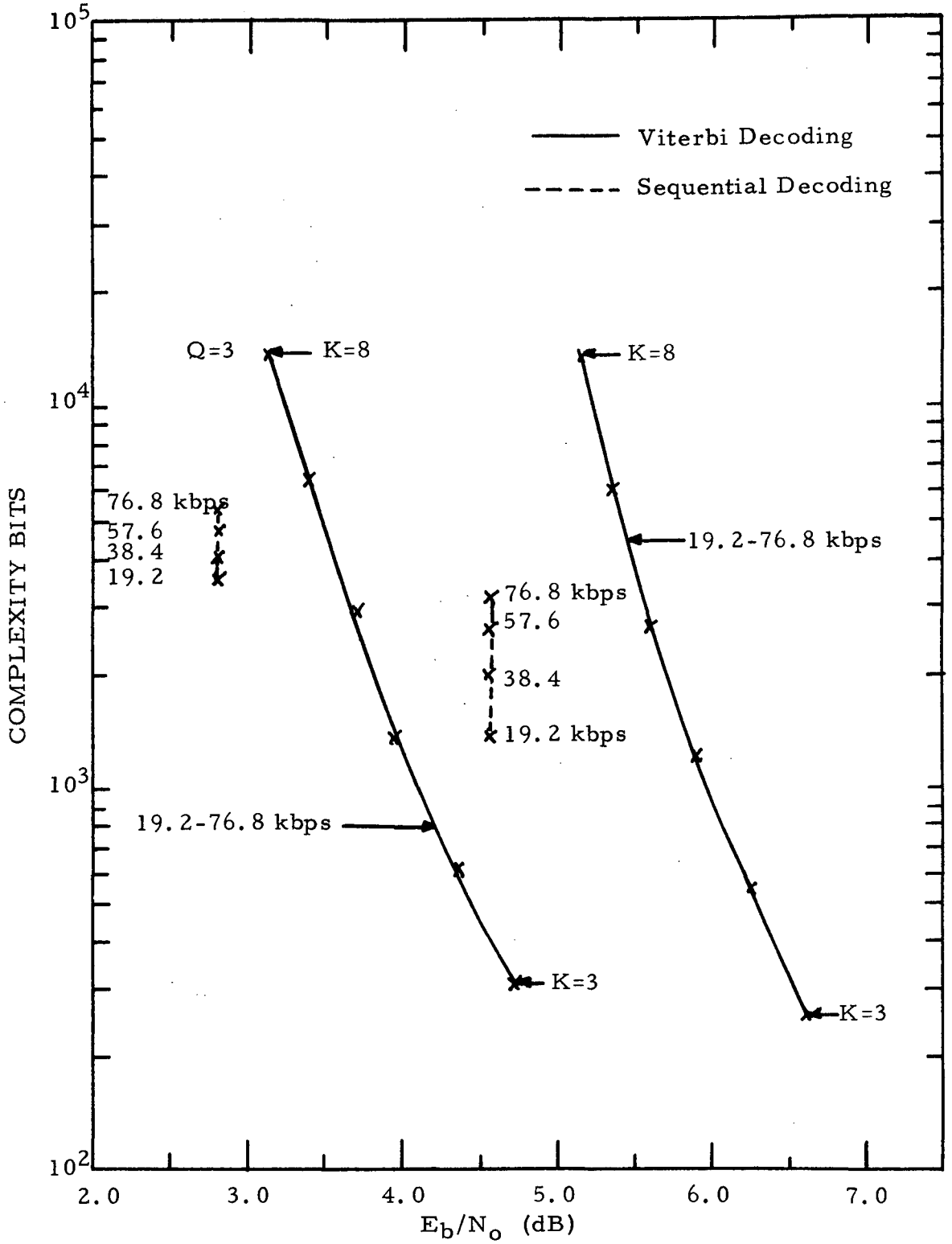


Figure 5.1 Comparison of Complexity Versus Performance of Viterbi Decoding and Sequential Decoding With Code Rate 1/2 and Output Probability of Error Per Bit of 10^{-4}

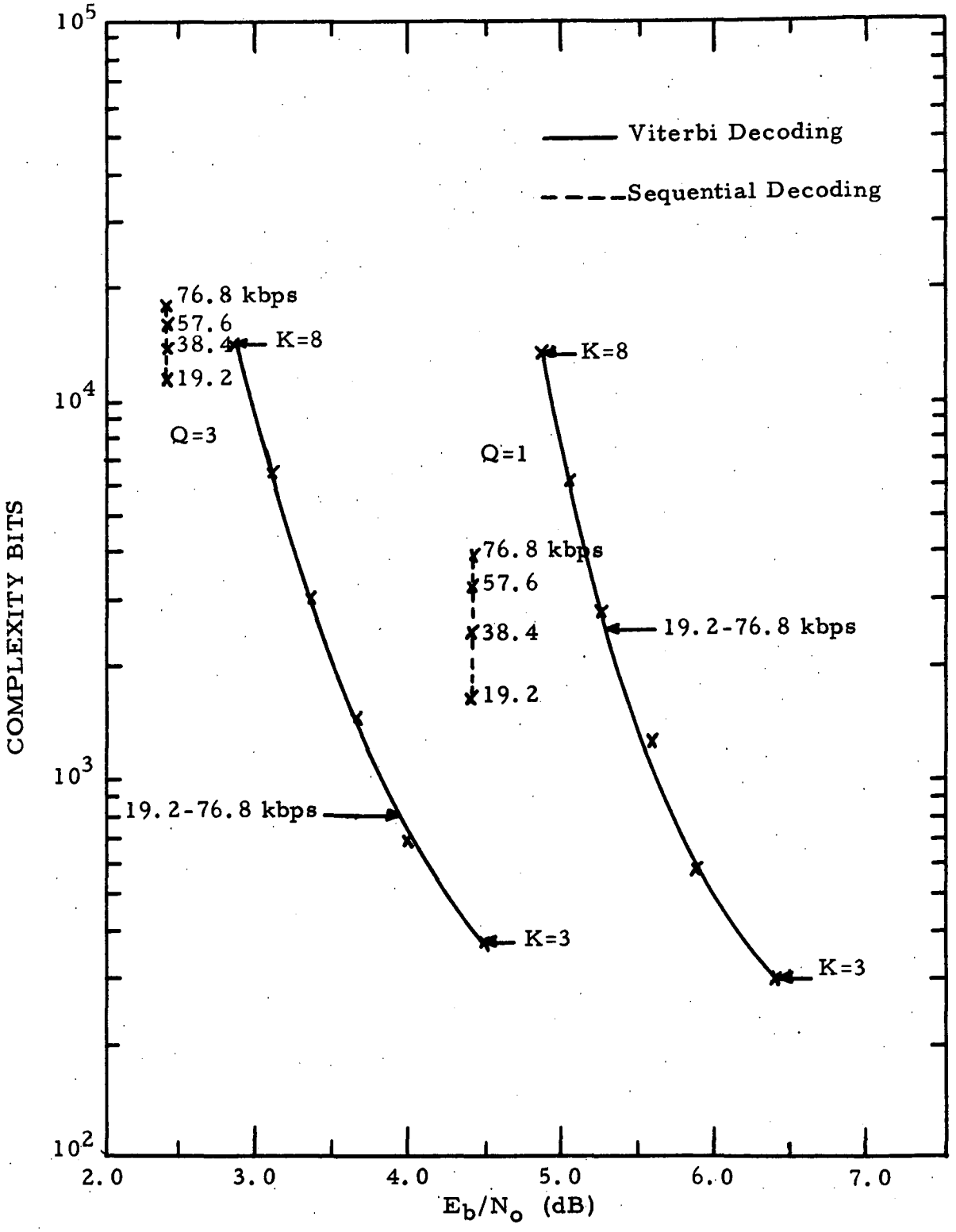


Figure 5.2 Comparison of Complexity Versus Performance of Viterbi Decoding and Sequential Decoding With Code Rate 1/3 and Output Probability of Error Per Bit of 10^{-4}

For the 76.8 kbps data rate and the same complexity, the sequential decoder requires 1 dB less E_b/N_o than the Viterbi decoder. Even for the high 9.1 Mbps data rate, the sequential decoder requires 0.5 dB less E_b/N_o to achieve an output probability of error per bit of 10^{-4} than the Viterbi decoder.

As the code rate is decreased and as the quantization on the demodulated signals is increased, the Viterbi decoder performance for a given complexity improves in relation to the sequential decoder since the sequential decoder must store the quantized demodulated symbols in its buffers while the Viterbi decoder does not. Even so, for rate 1/2 with three-bit quantization on the demodulated symbols, the sequential decoder requires 0.8 dB less E_b/N_o to obtain a 10^{-4} probability of error per bit than the Viterbi decoder for the same complexity at the 19.2 kbps data rate. Alternately, to achieve the same performance, the Viterbi decoder requires almost four times the complexity of the sequential decoder. For the 76.8 kbps data rate, the sequential decoder requires 0.65 dB less E_b/N_o than the Viterbi decoder of the same complexity or a Viterbi decoder must have three times the complexity to achieve the same performance as the sequential decoder. However, for the 9.1 Mbps data rate, the Viterbi decoder only requires half the complexity to achieve the same performance as the sequential decoder.

For rate 1/3 with three-bit quantization on the demodulated symbols, the sequential decoder still requires 0.5 dB less E_b/N_o to achieve a 10^{-4} probability of error per bit than the Viterbi decoder at the 19.2 kbps data rate. For rate 1/3 and Q = 1, the sequential decoder for data rate 19.2 to 76.8 kbps has an improvement in required E_b/N_o from 1.05 to 0.75 dB compared to the Viterbi decoder with the same complexity.

Finally, it is found that, for Q = 3, a sequential decoder with a code rate of 1/2 has an improvement in required E_b/N_o of 0.45 dB to 0.3 dB over a Viterbi decoder with a code rate 1/3 and the same complexity for data rates from 19.2 kbps to 76.8 kbps.

Decoding delay is of concern for some systems. However, the decoding delay for the sequential and Viterbi decoders is negligibly small even for the low 19.2 kbps data rate. In fact, the decoding delay for the sequential decoder at the 19.2 kbps data rate is only 19.8 milliseconds.

SECTION 6.0

AREAS FOR FURTHER STUDY

The present study has been concerned with the complexity and performance of channel coding (i.e., Viterbi and sequential decoding of convolutional codes) considering an ideal coherent PSK channel. However, to completely evaluate the performance of the channel coding in a system, the characteristics of the interface between the demodulation and the channel coding and of the interface between the channel coding and the data compression or source coding must be evaluated. Appendix II describes techniques to derive branch synchronization needed for Viterbi and sequential decoding and techniques to resolve reference phase ambiguity due to Costas loop suppressed carrier tracking for both biphase and quadriphase modulation. One technique for reference phase ambiguity resolution is the use of transparent codes as described in Sections 2.0 and 3.2. There is a need to study transparent codes to obtain "good" for the code memory lengths of interest for Viterbi and sequential decoding. Also, the branch synchronization techniques need to be investigated in order to obtain the performance of these techniques for good convolutional codes as a function of design parameters. The best method to study these areas is by computer simulation due to the large variety of codes and code memory lengths to be investigated.

Other areas of further study related to the demodulation-channel coding interface are the best choice of threshold spacing for three-bit quantization when quadriphase modulation is used, the sensitivity of AGC inaccuracies on Viterbi and sequential decoders, and the effects of the noise statistics out of the Costas tracking loop on the decoder. The last area, concerning the effects of Costas loop tracking noise statistics, has had very little study. Initial work was performed by Cahn, Huth and Moore[9] for the sequential decoder that identified some of the problems. The solution to handle the correlated noise statistics out of the Costas loop is to properly choose the branch metrics and convolutional codes that perform well with these statistics. Phase jitter of the modulator and doppler due to velocity and acceleration of the communication terminals should be examined for their effects on the Costas loop noise statistics and hence on the decoder. The best method for studying each of these areas is by computer simulation, since there are a large number of parameters to be varied.

In considering the interface between the channel coding and the source coding, the error burst distribution and the probability of error become important considerations, depending on the type of the source coding and the type of the source. For example, using scan-by-scan polynomial compression of a video signal, it is more desirable to have the errors occur in bursts rather than randomly. In this case, a burst of errors causes a loss of the scan but, with end of scan coding, other

scans are unaffected. Ideal codes for this type of source coding are convolutional codes since, when output errors occur, the errors occur in bursts.

In general, the more compression obtained by source coding, the higher the sensitivity of the source coding to errors and to error burst distribution. For these high compression techniques, the channel coding characteristics must be carefully considered. In some cases, pseudo-random interleaving is necessary between the channel coding and the source coding to provide a random distribution of errors. This increase in complexity must be considered in the overall system optimization, weighting the gains obtained by the channel and source coding. Thus, the requirements of the various source coding techniques in terms of data rate, allowed probability of error, and error burst distribution must be evaluated to determine the performance of the channel coding and source coding combination.

REFERENCES

1. Forney, G. D., Jr. "Coding System Design for Advanced Solar Missions," NASA Ames Research Center Final Report, Contract NAS 2-3637, Codex Corp., Watertown, Mass., December 1967.

2. Viterbi, A. J. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Transactions on Information Theory, Vol. IT-13, April 1967, pp. 260-269.

3. Viterbi, A. J. "Convolutional Codes and Their Performance in Communication Systems," IEEE Transactions on Communication Technology, Vol. COM-19, No. 5, Part II, October 1971, pp. 751-772.

4. Gilhousen, K. S., J. A. Heller, I. M. Jacobs, and A. J. Viterbi. "Coding Systems Study for High Data Rate Telemetry Links," NASA Ames Research Center Final Report, Contract NAS 2-6024, LINKABIT Corp., San Diego, Calif., January 1971.

5. Fano, R. M. "A Heuristic Discussion of Probabilistic Decoding," IEEE Transactions on Information Theory, Vol. IT-9, No. 2, April 1963, pp. 64-74.

6. Savage, J. E. "The Computation Problem with Sequential Decoding," Ph.D Thesis, Department of Electrical Engineering, MIT, February, 1965.

7. Huth, G. K. "Performance Bounds for Sequential Decoding Using Minimum Distance Properties," Ph.D. Dissertation, Department of Electrical Engineering, University of Southern California, September 1971.

8. Jacobs, I. M., and E. R. Berlekamp. "A Lower Bound to the Distribution of Computations for Sequential Decoding," IEEE Transactions on Information Theory, Vol. IT-13, April 1967, pp. 167-174.

9. Cahn, C. R., G. K. Huth, and C. R. Moore. "Simulations of Sequential Decoding with Phase-Locked Demodulation," IEEE Transactions on Communication Technology (submitted).

APPENDIX I

TRANSFER FUNCTIONS OF CONVOLUTIONAL CODES

The transfer function of a convolutional code is obtained by considering the modified state diagram of the code as a signal flow graph and using the general techniques discussed in numerous textbooks, such as Cheng,* and is summarized in the following.

An input variable is represented by a source node which has only outgoing branches; an output variable is represented by a sink node with only incoming branches. A path leading from a source node to a sink node without passing any node more than once is called an open path (also called a forward path).

In a general flow graph, there may be more than one open path between a source node and a sink node.

On the other hand, a feedback loop (or simply a loop) is a closed path. The loop transmittance of a feedback loop is defined as the product of the transmittances (or individual transfer functions) of the branches forming the loop.

The general formula for graph transmittance between a source node and a sink node in a signal flow graph is:

$$T = 1/\Delta \sum_i T_i \Delta_i \qquad \text{(I-1)}$$

---

*D. K. Cheng. Analysis of Linear Systems, Addison-Wesley, 1959.

where: Δ = 1 - (sum of all loop transmittances)

+ (sum of all products of loop transmittances of all possible nontouching feedback loops taken two at a time)

- (sum of all products of loop transmittances of all possible nontouching feedback loops taken three at a time)

+ ...

Δ_i = value of Δ for that part of the graph not touching the ith open path

T_i = path transmittance of the ith open path.

The summation is taken over all open paths between the source node and the sink node under consideration.

This general formula for a transfer function can now be applied to find the transfer function of the modified state diagram such as that shown in Figure 3.10. First, there are 11 feedback loops (loops) in the modified state diagram. These loops and their loop transmittances are:

$$\begin{aligned}
&(h) \quad L_1 = LND^2 && (bdhge) \quad L_7 = L^5N^3D^2 \\
&(cf) \quad L_2 = L^2ND^2 && (bcfdge) \quad L_8 = L^6N^3D^6 \\
&(bce) \quad L_3 = L^3ND^2 && (bdgfce) \quad L_9 = L^6N^3D^6 \\
&(fdg) \quad L_4 = L^3N^2D^4 && (bcfdhge) \quad L_{10} = L^7N^4D^4 \\
&(fdhg) \quad L_5 = L^4N^3D^2 && (bdhgfce) \quad L_{11} = L^7N^4D^4 \\
&(bdge) \quad L_6 = L^4N^2D^4
\end{aligned} \tag{I-2}$$

From these loop transmittances, Δ is found to be:

$$\Delta = 1 - \sum_{i=1}^{11} L_i + L_1(L_2+L_3+L_4+L_6+L_8+L_9) + L_2(L_6+L_7)$$

$$+ L_3(L_4+L_5) - (L_1L_2L_6 + L_1L_3L_4) \qquad \text{(I-3)}$$

By substituting the values of the loop transmittances into equation (I-3) and simplifying:

$$\Delta = 1 - LND^2(1 + L + L^2 + L^3N^2 - L^3N^2D^4 + L^4N^2 - L^4N^2D^4) \qquad \text{(I-4)}$$

There are seven (7) open path transmittances. These open paths with their transmittances T_i and Δ_i are:

(abce) $T_1 = L^4ND^6$

$$\Delta_1 = 1 - (L_1+L_4+L_5) + L_1L_4 \qquad \text{(I-5)}$$

$$\Delta_1 = 1 - LND^2(1+L^2ND^2+L^3N^2-L^3N^2D^4)$$

(abdge) $T_2 = L^5N^2D^8$

$$\Delta_2 = 1 - (L_1+L_2) + L_1L_2 \qquad \text{(I-6)}$$

$$\Delta_2 = 1 - LND^2 - L^2ND^2 + L^3N^2D^4$$

(abdhge) $T_3 = L^6N^3D^6$

$$\Delta_3 = 1 - L_2 = 1 - L^2ND^2 \qquad \text{(I-7)}$$

(abcfdge) $T_4 = L^7N^3D^{10}$

$$\Delta_4 = 1 - L_1 = 1 - LND^2 \qquad \text{(I-8)}$$

(abcfdhge) $T_5 = L^8N^4D^8$

$$\Delta_5 = 1 \qquad \text{(I-9)}$$

(abdgfce) $T_6 = L^7N^3D^{10}$

$$\Delta_6 = 1 - L_1 = 1 - LND^2 \qquad \text{(I-10)}$$

$$(abdhgfce) \quad T_7 = L^8N^4D^8$$

$$\Delta_7 = 1 \qquad \text{(I-11)}$$

By taking the summation over the open path transmittances and simplifying:

$$S = \sum_{i=1}^{7} T_i\Delta_i = L^4ND^6(1 + L^2N^2 - L^2N^2D^4) \qquad \text{(I-12)}$$

Hence, combining equations (I-1), (I-4), and (I-12):

$$T(L,N,D) = \frac{S}{\Delta} = \frac{L^4ND^6(1+L^2N^2-L^2N^2D^4)}{1-LND^2(1+L+L^2+L^3N^2-L^3N^2D^4+L^4N^2-L^4N^2D^4)}$$

$$= L^4ND^6(1+L^2N^2) + L^5N^2D^8\ 1+L+L^2(1+N^2) + 2L^3N^2$$

$$+ 2L^4N^2 + L^5N^4 + L^6N^4 + \ldots \qquad \text{(I-13)}$$

The infinite series obtained by dividing the numerator by the denominator enumerates the paths in the state diagram leaving the all-zeroes state and eventually returning to the all-zeroes state. From the series, as can be verified by examining the state diagram, there are two paths of weight 6, one of length 4 produced by a single input one, and one of length 6 produced by three input ones. There are no paths of weight 7, but there are ten paths of weight 8. The transfer function is, then, the closed form of the infinite series that defines the structure of the convolutional code.

APPENDIX II

TECHNIQUES OF BRANCH AND REFERENCE PHASE SYNCHRONIZATION

The interface problems between coherent PSK demodulation and the convolutional code decoder are the reference phase ambiguity and branch synchronization. These interface problems may be solved using either transparent codes and differential coding or nontransparent codes.

For biphase modulation, the incorrect reference phase could be corrected before the decoding by using differential coding after the encoding and before the decoding. However, since differential coding produces two errors for each single channel error, the effect of placing the differential coding before the decoder is to double the error rate into the decoder. The decoder normally operates where twice the symbol error rate into the decoder results in the loss of several dB in signal energy per bit/single-sided noise density (E_b/N_o). Using a transparent code and differential coding before the encoding and after the decoding, at most,* a double rate out of the decoder results. Since error rate is low at the output of the decoder, twice the error rate results in only a very small degradation; typically, at 10^{-4} error rate, less than 0.1 dB for sequential decoding and about 0.3 dB for Viterbi decoding with code memory length of K = 4.

---

* Simulations indicate a smaller loss due to clustered errors output from the Viterbi maximum likelihood decoder and the sequential decoder.

An incorrect reference phase can be corrected using a nontransparent code by detecting that no correct code word exists. This situation is detected by an increase in the growth rate of the best metric in the Viterbi decoder or an excess number of metric threshold releases in the sequential decoder. By using a simple up-down counter, the rate of best metric increase or the metric threshold releases can be measured. For the Viterbi decoder, each time a branch is received, the counter is counted down by one to a minimum of zero; the counter is counted up by N each time the metric accumulators are normalized. If the counter counts up to some threshold T, then an incorrect reference phase is declared and the incoming symbols are complemented. The parameters N and T determine the probability of detection, probability of false alarm, and the time to detect an incorrect reference phase. For sequential decoding, the counter counts up each time the metric threshold is released, and each time the metric threshold is tightened, the counter counts down to a minimum of zero. If the counter counts up to the maximum of the counter, then an incorrect reference phase is declared. The probability of detection, the probability of false alarm, and the time to detect an incorrect reference phase are determined from the size of the counter, the bias in the sequential decoder metric, and the metric threshold spacing.

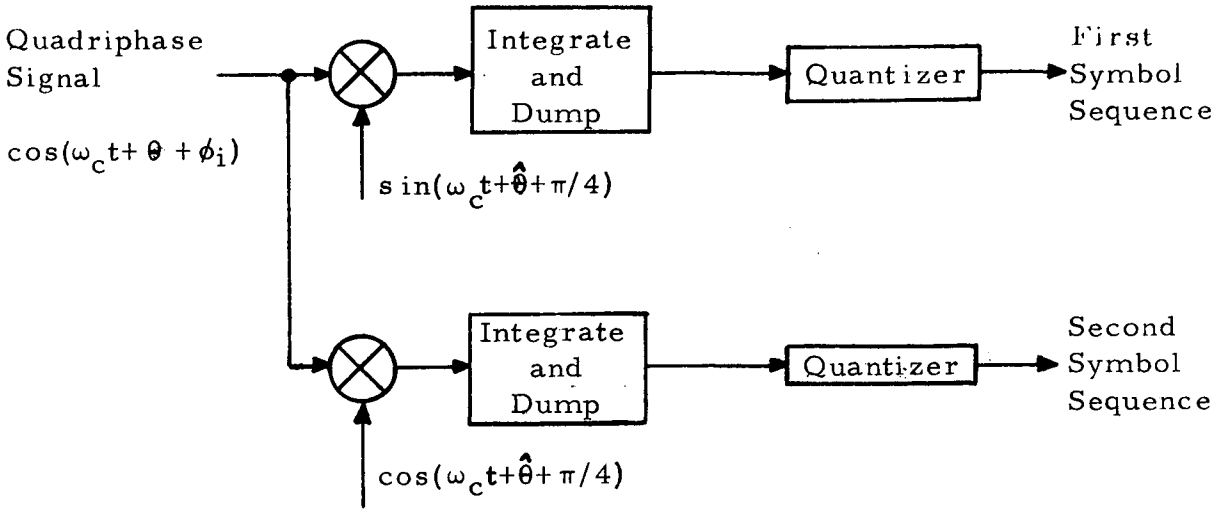
A branch for code rate 1/n is the n symbols associated with each information bit to be transmitted. Thus, branch synchronization is

required to establish which symbol in the received sequence is the first symbol on the branch, the second, etc. To determine branch synchronization with biphase modulation, the symbols in the received sequence can be arbitrarily assigned to branches. If the symbols do not belong together on a branch, then the decoder (Viterbi or sequential) can be used to detect that no correct code word exists with either a transparent code or a nontransparent code, similar to the resolution of phase ambiguity with a nontransparent code. Each of the n possible positions for the beginning of the branch is tested until the decoder indicates that a correct code word exists and branch synchronization is achieved. Since the probability of detection and the probability of false alarm would not be made one and zero, respectively, each of the n possible positions for the beginning of the branch may have to be tested more than once. Thus, the branch synchronization problem using biphase modulation can be solved in a straightforward manner, as can the phase ambiguity resolution problem for nontransparent codes.

For quadriphase modulation, there are three important cases to consider. The first case is a single binary source with a rate 1/2 convolutional code using each quadriphase digit as a branch. The second case is a single binary source with a rate 1/3 convolutional code. The final case consists of the use of the quadriphase modulation for multiplexing two binary sources.

Figure II.1 illustrates the quadriphase demodulation process and can be used as a guide in proposing solutions to the problems associated with each of the three cases to be considered. The quadriphase signal is $\cos(\omega_c t+\theta+\phi_i)$ where θ is a random variable due to channel noise and ϕ_i is the phase of the quadriphase modulation. The angle θ is estimated by a Costas loop or phase lock loop as $\hat{\theta}$. The reference signal $\cos(\omega_c t+\hat{\theta})$ is phase shifted to form $\cos(\omega_c t+\hat{\theta}+\pi/4)$ and $\sin(\omega_c t+\hat{\theta}+\pi/4)$. By mixing the quadriphase signal with these two signals and then integrating and quantizing, as shown in Figure II.1, the first and second symbol sequences can be separated. The vector diagram in figure II.1 illustrates the demodulation process as vector projections. The result of mixing the quadriphase signal with the cos and sin terms and then integrating over the symbol time is the projection onto the first and second symbol axes. Now, the projected values can be quantized as is done with biphase modulation with an appropriate scale factor.

In the first case to be considered, the single binary source with a rate 1/2 convolutional code using each quadriphase digit as a branch, quadriphase offers no further complication for the combination of branch synchronization and phase ambiguity than was encountered with biphase modulation. With quadriphase, there is a fourfold 90 degree phase ambiguity. However, this can be interpreted as a 180 degree phase ambiguity (i.e., the received sequence is complemented) and a $\pm$ 90



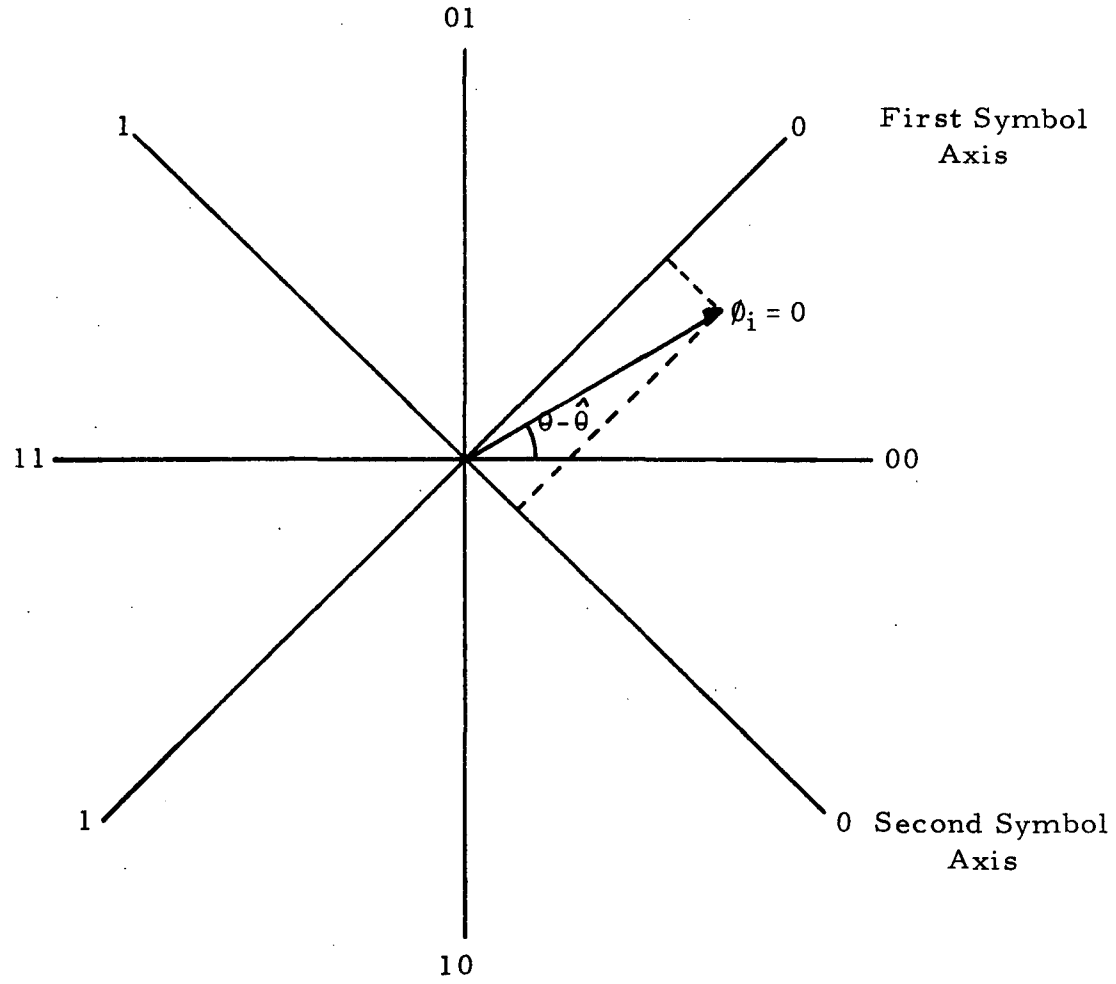


Figure II.1 Quadriphase Demodulation

degree phase ambiguity. Thus, the 180 degree phase ambiguity can be solved by using a transparent code and differential coding on the binary source. The $\pm$ 90 degree phase ambiguity can be resolved in the same way as branch synchronization for biphase modulation (i.e., detect that no correct code word exists and try complementing the first symbol sequence while interchanging the two symbol sequences). Therefore, this case is solved by the techniques and design parameters found for the biphase modulation case.

In the second case to be considered, the single binary source with a rate 1/3 convolutional code, both branch synchronization and a fourfold 90 degree phase ambiguity must be resolved. It is possible, again, to interpret the fourfold 90 degree phase ambiguity as 180 degree phase ambiguity and a $\pm$ 90 degree phase ambiguity, resolving the 180 degree phase ambiguity with a transparent code and differential coding on the binary source. However, the $\pm$ 90 degree phase ambiguity and a threefold branch synchronization ambiguity must still be resolved. Thus, if it is detected that no correct code word exists, then each of the six (6) possible choices for the branch synchronization and phase reference (i.e., three branch beginning points for each of the two possible phases) must be tested. Using a nontransparent code would require twelve (12) possible choices for branch synchronization and phase reference, so the performance of transparent codes with differential coding is a very important consideration.

In the final case to be considered, two binary sources multiplexed by the quadriphase modulation, transparent convolutional codes and differential coding can be used to reduce the fourfold 90 degree phase ambiguity to an ambiguity of which sequence is the first symbol sequence. This possibility is due to the two independent sources (possibly of different data rates) being convolutional encoded separately before the modulator. Thus, a 190 degree reference phase shift complements both sequences and has no effect, while a $\pm$ 90 degree reference phase shift interchanges the symbol sequences and complements one of the sequences, but the complementing of either sequence has no effect. In order for the decoder to detect the wrong sequence is being sent to it, either the data rates of the two sources must be different, the code rates must be different, or the convolutional codes used to encode the two sources must be different and carefully chosen. To resolve the branch synchronization of both decoders of the binary sources simultaneously would require testing a number of possibilities equal to the least common multiple of the symbol rates. For many combinations of sources, the branch synchronization problem of the two decoders becomes so difficult (a large number of possibilities to be tested) that a transmitted synchronization sequence is the least complex. However, the use of transparent codes has again reduced the ambiguity to be resolved by a factor of two.